ALGORITHMS FOR STREAMING, CACHING AND STORAGE OF DIGITAL MEDIA

by

Zhourong Miao

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(ELECTRICAL ENGINEERING)

February 2002

Contents

List of Tables iii

List of Figures iv

Abstract vi

1 Introduction and Motivation 1
- 1.1 Digital communication and multimedia data delivery . . . 1
  - 1.1.1 Focus of this thesis . . . 4
- 1.2 VBR video characteristics . . . 6
  - 1.2.1 VBR video . . . 6
- 1.3 Real-time VBR video delivery and its challenges . . . 11
  - 1.3.1 Delivery over best-effort networks . . . 11
  - 1.3.2 Joint approaches . . . 12
  - 1.3.3 Rate control and error control . . . 13
  - 1.3.4 Delivery over QoS networks . . . 17
- 1.4 Other Components in streaming video services system . . . 18
  - 1.4.1 Proxy caching for streaming video . . . 18
  - 1.4.2 Disk storage strategy for central video server . . . 20
- 1.5 Contributions of this thesis . . . 22

2 Scalable streaming media delivery 24
- 2.1 Introduction . . . 24
- 2.2 Scalable Media and Streaming System Architecture . . . 29
  - 2.2.1 Data packetization . . . 29
  - 2.2.2 System architecture . . . 32
- 2.3 The Packet Scheduling Problem . . . 34
- 2.4 Proposed Scheduling Algorithm . . . 36
  - 2.4.1 Expected run-time packet distortion . . . 37
  - 2.4.2 Expected Run-time Distortion Based Scheduling – ERDBS . . . 41
  - 2.4.3 Discussion . . . 41
- 2.5 Experimental Results . . . 42
- 2.6 Conclusions . . . 44

3 Scalable Proxy Caching for Streaming Video Applications 46
3.1 Introduction . . . 46
3.1.1 System architecture . . . 48
3.1.2 Related work . . . 52
3.2 Basic definitions . . . 55
3.3 Video caching in QoS networks . . . 57
3.3.1 Problem formulation . . . 57
3.3.2 Analysis on bandwidth reduction . . . 58
3.3.3 Client buffer analysis . . . 61
3.3.4 Proposed SCQ algorithm . . . 64
3.4 Video caching in best-effort networks . . . 66
3.4.1 Problem formulation . . . 69
3.4.2 Analysis on buffer trace after caching . . . 70
3.4.3 Proposed SCB algorithm . . . 72
3.4.4 Caching table . . . 74
3.5 Experimental results . . . 74
3.6 Conclusions . . . 77

4 Video Compression with Rate Control for Video Storage on Disk Based Video Servers 88
4.1 Introduction . . . 88
4.2 Problem Formulation . . . 94
4.3 Optimization based on Multiple Lagrange Multipliers . . . 100
4.4 Experimental Results . . . 102
4.5 Conclusions . . . 105

5 Conclusions 107

Reference List 109

List of Tables

4.1 Notation used in this chapter 95

List of Figures

1.1 Streaming video system examples 4
1.2 The relationship between the topics in this thesis and streaming video applications. 5

2.1 (a) MPEG 4 FGS packetization of a single frame. (b) MPEG 4 FGS packetization of frames with inter-frame dependencies. 30
2.2 System architecture 32
2.3 MPEG 4 FGS frame data rate. The dotted line is the base layer size of each frame; and the solid line represents the full size of each frame (base layer and enhancement layer). 43
2.4 The comparison of playback quality (PSNR) using proposed ERDBS and SS algorithm, in various conditions, such as bandwidth, channel packet loss rate, start-up delay and round trip time. In most cases the playback quality of ERDBS outperforms the regular SS delivery algorithm around 2 dB. 45

3.1 System architecture. The proxies are set close the clients, and are connected to the video server via either a QoS (a) or best-effort network (b). 48
3.2 Cumulative rate and slope functions. (a): Cumulative frame/channel rate. Curve (1), (2) and (4) are cumulative channel rate functions, $\sum_i C(i)$. Among them only (1) and (4) represent CBR channels. Curve (3) is the cumulative frame rate, $S_{\mathcal{A}}(t)$. Curve (1) and (2) are feasible channel rate, while (4) is not. (b): Slope function $L_{\mathcal{A}}(t)$ is drawn in curve (3). The minimum CBR channel bandwidth that has to be reserved is $C_r = \max\{L_{\mathcal{A}}(t)\}$. 60
3.3 Illustration of Proposition 1. The lower figure shows the slope functions before and after caching one frame $F(k_1)$, represented by $L_{\mathcal{A}}(t)$ (solid line) and $L_{\mathcal{A}_1}(t)$ (dashed line), respectively. The upper figure is the corresponding cumulative channel/frame rate functions. After caching an "earlier" frame before t_{peak}, i.e., $k_1 \leq t_{peak}$, $\max\{L_{\mathcal{A}}(t)\}$ reduces from c to c_1. Obviously, caching a "later" frame $F(k_2)$, i.e., $k_2 > t_{peak}$, can not reduce $\max\{L_{\mathcal{A}}(t)\}$, which occurs at t_{peak}. 82

3.4 Illustration of caching one frame before or after t_{max}. $S_{\mathcal{A}_1}(t)$ and $S_{\mathcal{A}_2}(t)$ are the cumulative frame rate functions after caching $F(k_1)$ and $F(k_2)$, (drawn in dashed and dotted lines) respectively. Note that only *one* frame is selected in each case. If $R(k_1) = R(k_2)$ and $k_2 < t_{max} < k_1 < t_{peak}$, Proposition 1 shows that selecting $F(k_1)$ or $F(k_2)$ leads to the same reduction on bandwidth C_r. However, the corresponding changes in B_{max} are different: caching $F(k_1)$ reduces B_{max} from b to b_1 ($\Delta b = b_1 - b < 0$); while caching $F(k_2)$ increases B_{max} from b to b_2 ($\Delta b = b_2 - b > 0$). Therefore caching a frame between t_{max} and t_{peak} (e.g., $F(k_2)$) can reduce B_{max} while keeping the same reduction on C_r. 83
3.5 Trace of client buffer size in number of frames and bits. 84
3.6 (a): $B_\phi(t)$, no frame is cached. (b): $B_{\mathcal{A}_1}(t)$, frame $F(t_1)$ is cached. These figures illustrated that by caching one frame $F(t_1)$, the buffer trace can be "lifted" for $t \geq t_1$, when there is no buffer size limitation. 84
3.7 The client buffer size limitation is B_{MAX}. (a): $B_\phi(t)$, no frame is cached. (b): $B_{\mathcal{A}_1}(t)$, after $F(t_1)$ is cached. (c): $B_{\mathcal{A}_2}(t)$, after both $F(t_1)$ and $F(t_2)$ are cached. With the buffer size limitation, the buffer trace after caching frames will not follow that in Fig. 3.6. (b) shows that caching frame $F(t_1)$ only increase $B_{\mathcal{A}_1}(t)$ between $t_1 \leq t \leq t_{max}$. (c) shows that caching frame $F(t_2)$ can lift buffer trace for all $t \geq t_2$ because there is no maxima point after t_2. 85
3.8 (a): Trace where only I_{req} is cached and troughs occur at time t_1 and t_4. (b): After SCQ caching. First select frames before t_1, but t_4 still remains the same, due to the maximum peak at t_2, drawn in dotted line. Next select frames within $[t_2, t_4]$ to increase robustness for t_4. . 85
3.9 (a) The bandwidth ($C_r(\mathcal{A})$, see (3.7)) that has to be reserved, v.s. the percentage of the video been cached. (a): Both the proposed SCQ and prefix caching can reduce $C_r(\mathcal{A})$ similarly as more portion of the video has been cached. (b): The maximum buffer size, B_{max}, required at the client to achieve the caching performance in (a). 86
3.10 Robustness vs the percentage of the video been cached, using SCB and prefix caching methods. (a) Robustness U defined in (3.17). (b) Robustness U_a defined in (3.18). 86
3.11 Robustness verification, with 1000 realizations used in each case. (a): T_J/T_V vs percentage of the video being cached. (b): T_J/T_V vs channel error ϵ, when 20% of the video is cached. (c): T_J/T_V vs average channel congestion duration d_c. 87

4.1 Service Round. During each service round time, T_{round}, many users can retrieve data from the server for playback. This figure shows an example of Round Robin Service, where each user is allowed to retrieve a block data from a particular time slot during each service round. 90
4.2 Disk placement. The video blocks are placed in an "interlaced" fashion so that disk seek-time can be reduced when multiple video objects are requested concurrently 92
4.3 Accumulated channel rate and frame rate. 98
4.4 Accumulated channel rate and frame rate 103
4.5 PSNR of the video sequence. When the maximum number of the concurrent user can be supported by the server increases, the bandwidth allocated to each user decreases, therefore the video bit rate has to be reduced which leads to poor quality. Using rate control based compression can improve the overall PSNR (averaging over the frames) by 0.5 to 1.5 dB compared to the scheme without any rate control (a uniform quantizer is used for all frames). 104
4.6 Average waiting time and hiccups without rate control 105

Abstract

Streaming media applications have become important components in multimedia communications. Typically, these applications require real-time data delivery in order to provide continuous playback with good visual quality. However, the real-time constraints may not be explicitly guaranteed when the streaming media is delivered over networks exhibiting time varying behavior. Streaming systems are designed to maximize the playback quality in the presence of various channel conditions. This research includes studies of different components of a streaming system, and proposes several algorithms to improve the streaming performance.

The first part of this thesis considers the transport of scalable streaming media over best-effort networks (e.g., today's Internet) and proposes a scheduling algorithm for packet delivery. The proposed algorithm first determines the importance values of all packets in the transmission buffer, based on the packet contents, channel conditions and client feedbacks. Then the algorithm guides the media server to transmit more important data packets earlier than less important ones. This leads to improvements in the playback quality.

The second part focuses on video caching, and shows that the streaming performance can be improved even when only part of the video object is cached in the proxy. Two video caching algorithms are proposed to store selected frames of the video in the proxy, under the constraints of cache space and decoder buffer size. The first caching algorithm aims at reducing the cost for channel bandwidth reservation in QoS networks; while the second one is designed for best-effort networks, with the goal to improve the robustness of continuous playback against poor channel conditions (e.g., packet delay and loss).

The last part of this thesis addresses the video compression problem combined with disk storage strategies for video servers. Video disk storage algorithms aim at improving the video server throughput by placing the video data blocks in a special order (therefore reducing the disk seek time). We translate the specified disk placement algorithm into rate constraints for video compression, and propose a rate-distortion based compression algorithm to improve the video quality, while maximizing the advantage achieved with the disk placement strategies.

Chapter 1

Introduction and Motivation

1.1 Digital communication and multimedia data delivery

The development of high speed networks has led to a steady increase in the popularity of digital multimedia communications. Multimedia applications are becoming an important component for today's Internet, from commercial services, such as online video/audio clips, video conferencing, video-on-demand, to distance learning, tele-presentations, digital libraries, and other scientific and medical research, such as remote surgery, etc..

Streaming media data usually has the following properties: (i) the client can play back the media before it is fully downloaded, with a fixed start-up delay. Typically, all the data packets have to arrive at the client end before their playback time. Therefore, this delay constraint requires sufficient channel bandwidth to deliver the

streaming data on time. (ii) Using retransmission can not always recover all the lost/corrupted packet due to the delay constraints. (iii) The media data packets may have different impacts on the quality of the reconstructed signal and therefore the loss of some (important) packets may have a more severe impact on quality than other packets.

For such streaming media applications, Quality-of-Service (QoS) guarantees in the network are very useful, including for example, constant delay, sufficient bandwidth and low data loss rate during playback. Therefore, transporting streaming media over QoS networks can easily achieve better performance. An example of a QoS networks considered in this thesis (Chapter 3) is a network based on Asynchronous Transfer Mode (ATM) [23]. The main issues for delivery of streaming media over the QoS networks can be, for example, admission control, channel utilization, bandwidth cost etc.

However these QoS parameters can not be easily provided by best-effort networks, e.g., the Internet. Best-effort networks are suitable for delivering bulk data with relatively loose timing requirements (e.g., text, emails, data files). Thus, the channel packet losses can be recovered by retransmissions. But due to its extra delay, retransmission can only recover a limited part of the lost packets in real-time streaming applications. Consequently, it becomes a challenging issue in streaming media applications to provide robust transmission and therefore better playback quality against delay, congestion and packet loss, especially in best-effort networks.

As the Internet grows and its costs come down, best-effort networks represent the majority of practical networks used today. The delivery of streaming media over the Internet attracts more interest, from academic research to industrial development. Though there have already been some successful commercial products for streaming applications over the Internet, those challenging problems, such as delay, jitter, congestion, packet loss, still remain open.

The increase in Internet data traffic also leads to the quick development of proxy based caching in recent years. The initial research in this area (e.g., within the Harvest project [11]) has led to the development of commercial products (e.g., [25]) and to continuing research activity (e.g., [87, 96]). Due to the increasing demand of streaming video service and its large volume of data traffic, proxy caching of streaming video (as well as other multimedia objects such as images, audio clips) is becoming increasingly important.

Similar to the video end-to-end delivery methods, caching strategies specific to video are often designed differently from other methods for caching "traditional" web objects. Studies in [101, 50, 84] show that the benefits from video caching include not only the reduction in networks access cost and delay, but also an improvement in the overall performance of streaming video applications, e.g., more robust packet delivery against poor network conditions.

Examples of video streaming systems are shown in Fig.-1.1. The encoded video objects are stored on the video server (live video is encoded in real-time). The servers

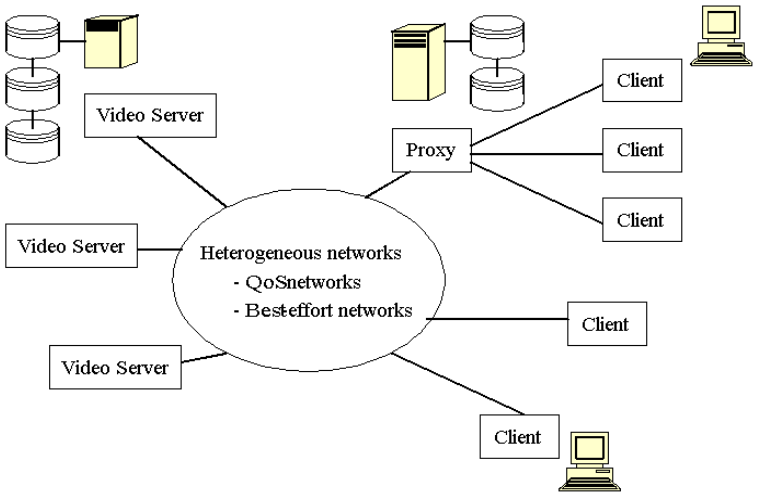

Figure 1.1: Streaming video system examples

and clients are connected by some heterogeneous networks, e.g. they can be either QoS networks or best-effort networks, and may have various bandwidth capacities and packet delays. Some clients can be directly connected to the servers (through the communication networks), while others may be connected to them through the proxies, where the proxies are set close to those clients and connect to the servers through the communication networks. Here the proxies are expected to improve the performance of streaming video services.

1.1.1 Focus of this thesis

This thesis includes the studies on three topics, each focusing on a different component of the streaming video system shown in Fig.-1.1, namely, (i) the video transmission problem (Chapter 2); (ii) the video proxy caching problems for QoS and best-effort networks (Chapter 3), and (iii) video compression constrained by the

server storage restrictions. (Chapter 4). The relationship between each topic and the overall system is further depicted in Fig.-1.2.

The streaming media objects can be video, audio, speech, graphics or any combination of them. Most of this thesis addresses variable-bit-rate (VBR) video objects, which usually have high data volume and bit rate burstiness. However, the proposed algorithms, for example the delivery scheduling in Chapter 2, can also be applied to other media objects, e.g., the audio clips.

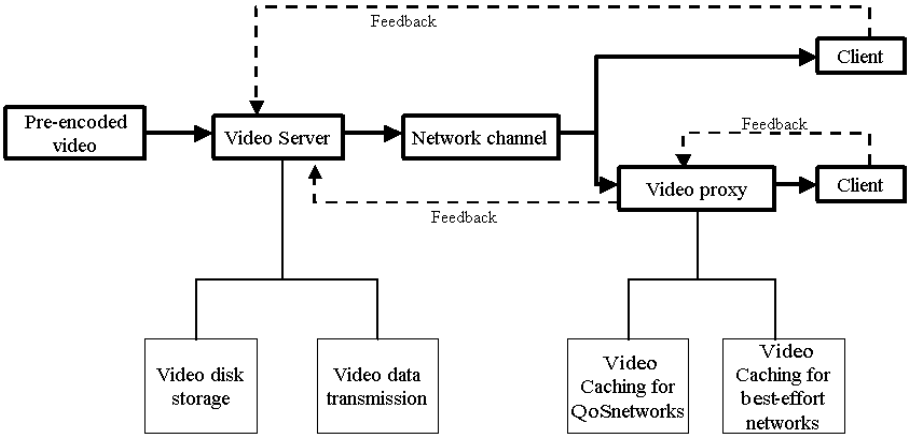

Figure 1.2: The relationship between the topics in this thesis and streaming video applications.

In the rest of this chapter, we will first describe the characteristics of VBR video, then we will give brief reviews on the related areas of streaming video delivery, caching and disk storage. The contributions of the thesis on each topic are summarized at the end of this chapter.

1.2 VBR video characteristics

In this section, we review some properties for streaming media objects and their delivery challenges, especially for Variable-Bit-Rate (VBR) video.

1.2.1 VBR video

In order to achieve acceptable visual quality, video data is typically large even after compression. For some good quality video stream, e.g., compressed digital movies, the data volume can be huge and can pose significant challenges to system resources, such as server disk storage, disk bandwidth, network channel bandwidth, codec buffer size, etc. For example, a 2 hour movie may have a total size of 4 gigabytes. The compressed video signal usually produces a variable bit rate stream, i.e., some frames (e.g, frames containing "complex" visual objects and/or more motions of the objects) may be coded with more bits than other frames (e.g, with simple objects and less motion). This phenomenon is also referred to as data *burstiness* of the VBR video. The burstiness introduces difficulties for video transmission over a constant-bit-rate (CBR) channel, and has been studied in some literature as well as in this thesis (see Chapter 3 on video caching).

1.2.1.1 Rate-distortion model

Most current coding standards for images, video and audio, such as JPEG [63], MPEG [52], H.263 [34], use lossy compression techniques to achieve high compression gain, in which the detail information of the source may discarded, while the reconstructed signal still has reasonable perceptual quality. The amount of output data from lossy compression can be increased/decreased by increasing/decreasing the quality of the reconstructed signal.

The fundamental mathematical principle behind this property is the "rate-distortion" theory. The classical rate distortion (R-D) theory has its origins in Shannon's theory [86], which is concerned with the task of representing a source with the minimum number of bits possible, subject to a fidelity criterion.

R-D based video compression for different applications has been studied extensively in recent years, [82, 90, 57]. The basic idea in most R-D based approaches is to properly allocate the amount of bits to different parts of the signal for a given a rate budget (e.g., total number of bits for the compressed signal), so that the quality of coded signal is maximized. For example, video objects of real-time delivery are usually compressed with a given target rate, based on the available channel bandwidth. Under this rate constraint, the compressed video quality can be maximized by using R-D based compression approaches [32].

Note that because the channel rate may be unknown in the compression stage due to heterogeneity of the network, in some cases it is preferable to compress the

video at high quality with high bitrate, and let the sender application decide the number of bits to be transmitted during the delivery. This highlights the usefulness of the so-called *scalable* or *layered* encoded video (similar scalable formats can be found in still image or audio coding). In fact scalable coded video also has other advantages over non-scalable systems, which will be discussed in the next section.

1.2.1.2 Data contents

One important property of compressed multimedia data is that different parts of the bitstream have different "importance value" for the quality of reconstructed signal. There are different ways to classify the compressed data into categories with different "importance". For example, in transform-based (DCT or wavelet) still image compression, the low frequency coefficients are treated as more important data than high frequency coefficients, because low frequency coefficients can be used to reconstruct a coarse version of the signal, and high frequency coefficients can be decoded to obtain additional fine details based on that coarse version. The different importance among those coefficients can also be found in video compression, where some frames are coded as still images (e.g., the I-frames coded in *intra* mode). Moreover, in compressed video data, "data dependencies" also contributes to producing different levels of data importance. For example, I-frames are more important than P-frames (coded in *predictive* mode), because P-frames are coded depending on the data in corresponding I-frames (P-frames can not be decoded without the presence of I-frames) [3].

Thus, in the presence of limited channel bandwidth resources, or when network congestion occurs, the less important data can be discarded by the sender or the intermediate nodes in the networks, in order to give more chance to the more important data to be delivered on time. To better utilize this property, one may choose to use a *scalable* encoding scheme.

1.2.1.3 Scalable and non-scalable representations

Scalable (or layered) compression techniques are attractive because of the additional flexibility and functionality they provide, although this comes at the cost of a reduced compression performance. In a scalable coding scheme, the original signal is coded into several *layers*, from lowest to highest layer. The lowest layer, (also referred as the *base layer*), contains a coarse version of the signal, each of the higher layers (referred as *enhancement layers*) carries finer information of the original signal. To reconstruct the signal, one can decode starting from base layer to an arbitrary higher layer. The more layers are decoded, the better the quality of the reconstructed signal is. Note that in order to decode a higher layer, *all* the layers lower then that particular higher layer must be decoded first. In other words, a higher layer is useless without the presence of all corresponding lower layers.

Layered coding for image compression has been used in several systems, e.g., progressive compression in JPEG standard. The new JPEG 2000 standard supports still image compression with a better embedded scalability (by using wavelet transform based compression) [1].

Scalable video is also supported by some standards, such as H.263+ and MPEG-4 [3]. MPEG-4 specifies a scalable coding format for video, referred as fine-granular-scalability (FGS), which encodes video into two bit streams, i.e., the base layer and enhancement layer. The base layer has to be decoded completely and then the enhancement layer can be decoded to an arbitrary position [65].

While, typically scalable video encoding produces less compression gain as compared to non-scalable techniques, it is still preferred in some situations for its advantages over non-scalable video. For example, to access versions of the video with different qualities, several complete bit streams of compressed video have to be stored by using non-scalable coding. In contrast, only a *single* scalable video bit stream is needed in that case, so that the disk storage can be reduced.

For streaming video *multicast* applications, some end users connected with channels having small bandwidth may prefer a low bitrate video stream with poor quality, while some users with a high capacity channel may prefer a higher bitrate video with better quality. This conflict can be easily solved by sending a *single* scalable video bit stream with full quality, so that the users can decide to receive a number of layers that matches their channel capacity. This approach is known as receiver driven multicast [47, 48, 18].

For end-to-end unicast streaming video, scalable video offers better flexibility for different network conditions, e.g., when the network suffers from congestion or delay, the sender can send fewer number of layers to maintain the continuous playback

at receiver; conversely the number of layers can be increased when the network conditions become better.

Chapter 2 of this thesis studies the transmission strategy for end-to-end scalable streaming video. By analyzing and exploring the flexibility of the scalable video, a transmission algorithm is proposed to achieve better receiver playback quality. Chapter 3 and 4 do not explicitly specify the video format, the algorithms proposed in those chapters are applicable to both scalable and non-scalable videos.

1.3 Real-time VBR video delivery and its challenges

1.3.1 Delivery over best-effort networks

Delivery of streaming media may requiring some QoS guarantees on channel bandwidth, delay and loss rate that may not be explicitly available in best-effort networks.

Delay. Real-time streaming applications require strict delay in packet delivery, since those packets that arrive to the client later than their playback time are considered to be useless. This delay bound also introduces the following requirements for the channel: bandwidth and packet loss.

Bandwidth. The channel should have enough bandwidth to deliver the packets on time, especially for video data which is typically large.

Loss. Since retransmission introduces extra delays, it can not recover all the lost packets in real-time streaming applications. Therefore the channel should have low loss rate, or have extra bandwidth to deliver additional error resilient data.

Usually in best-effort networks, these parameters can not be easily controlled by the end users (sender or receiver), and the network behavior may be difficult to predict. Since these parameters are time varying, a common mechanism uses network feedback to estimate them during the transmission. The lack of these QoS guarantees is a very significant challenge in designing an efficient streaming video system.

1.3.2 Joint approaches

Approaches to improve the performance of Internet streaming video can be roughly classified into two general categories: *rate control* and *error control* methods. Before we give an overview of these approaches, it should be pointed out that most of these approaches are designed in a *joint* fashion by considering the characteristics of *both* the media data contents and the network conditions. The main reason for the better performance of "joint" approaches is that the compressed media data has different levels of importance (and data dependencies), as mentioned above. Joint approaches can treat the data packets differently according to their importance to achieve better overall performance.

For example, from a coding point of view, Joint Source Channel Coding (JSCC) has been proven to have advantages in coding multimedia signals with protection against channel error (for example, unequal protection coding, multiple description coding, etc..)

From a network point of view, since the traditional protocols (such as TCP/UDP, which treat all data packets in the same way) are not suitable for real time transmission, the new protocols proposed (such as RTP/RTSP [4]) are designed to be more specifically for streaming media delivery. There are other proposed delivery mechanisms that consider both the content of data packets and channel conditions, for example, priority packet transmission or dropping, [64].

Even in other components of the system, such as the proxy caching and central server disk storage, the joint approaches can lead to attractive performance improvements. In this thesis, the proposed strategies in different topics (delivery, caching, disk storage) also have the "flavor" of a joint approach, and will be shown to have improved performance for streaming applications (covered in Chapter 2-4).

1.3.3 Rate control and error control

1.3.3.1 Rate control

The heavy data traffic of streaming video may introduce network congestion (of course, the congestion can be also due to other reasons such as physical problems of internal network nodes), which in turn, causes packet losses and degrades the

quality of the playback. "Rate control" methods are commonly applied to solve this problem, by considering the channel condition as a critical parameter during the data encoding procedure. To obtain the channel conditions, one can use a predefined channel model or estimate it from the real time network statistics based on the feedback; or combine them both, e.g., use a priori channel model and update it during the data transmission according the network status.

The rate control algorithms can be roughly divided into the following two categories.

1. Encoder rate control. This approach operates directly in the video compression domain, which basically reduces the output bitrate from the encoder when network congestion occurs, and maintains the output bitrate under the available channel bandwidth when there is no congestion.

2. Network interface rate control (rate shaping/smoothing). Instead of varying the actual bits of compressed frame, it controls the video data to be fed into to the channel, or statistically multiplexes multiple streams into the channel, to maximally utilize the channel bandwidth.

1.3.3.2 Error control

For data (such as text, files) with lax timing requirement, error control is easy to achieve, e.g., retransmission can be simply applied to recover lost data. Due to the delay constraints, streaming applications have to use more carefully designed error

control mechanisms to achieve better performance. Some approaches are listed as follows.

1. Forward error correction (FEC). Both compression and error protection are applied during the encoding of original signal and the transmission of coded data. Joint Source Channel Coding (JSCC) has been studied extensively ([26, 36]) in designing of FEC approaches. FEC is applied in the situations where the network feedback is difficult to obtain (to guide the retransmission), or the round-trip-time is too long for retransmission. For example in *multicast* streaming video, FEC is preferred to avoid feedback explosion and improve channel utilization, [18].

2. Error resilient coding. This approach has some similarity with the FEC approach. However error resilient coding operates mostly in the signal compression stage. Different compression methods, with different error resilient capability, are chosen according to the channel environments. For example, Multiple Description Coding (MDC) is shown to be more robust against channel noise than layered coding for audio or video in certain cases, i.e., in the case when there is no prioritized delivery (which is common for the Internet), or packet retransmission is not allowed (due to tight delay constraint) or not provided (e.g., lack of feedback channel) [35, 89]. Another example can be the "compression mode selection" (intra/inter mode) in video compression [27], where the more robust compression mode (intra) is chosen when the channel

noise is high, at the cost of requiring a higher transmission rate than if inter coding modes were used.

3. Retransmission. Though being tightly restricted by the delay bound, retransmission can be applied with certain limitation in streaming media applications. The "limitation" means that (i) the retransmission is only useful if the packets can still arrive the client before time out, (ii) when there is only limited channel bandwidth, e.g. the "overall" transmission rate is limited, a retransmission of one lost packet may reduce the chances for transmission of other packets (or transmission of other lost packets). The sender has to make decisions on whether or not to abandon the retransmission requests in order to save bandwidth for other packets. There have been works on retransmission-based error control for both unicast [24, 59] and multicast [18, 62, 41, 97].

The retransmission issue leads to an interesting media delivery problem: *how to deliver (considering (re)transmission of both new and lost packets) a given media stream over a lossy channel with delay constraint?* In fact studies have shown that to achieve better performance, not only the decisions on packets retransmission should be carefully made, but also the delivery *schedule* for both new and lost packets is important to improve the playback quality, e.g., see [64].

1.3.3.3 The media data delivery problem

In Chapter 2 of this thesis, a general framework of scalable streaming media delivery problem is presented. This study focuses on the end-to-end unicast streaming applications, using a scalable (layered) encoded media. Each data packet may contain data of different layers and frames, thus has different importance "value" for the playback quality. The channels in best-effort networks between sender and receiver have certain loss rate and delay, which can vary over the time. The streaming media data packets are transmitted via the forward channel from sender to receiver; while the packet loss (as well as other statistics such as RTT) are reported to the server via a feedback channel. Chapter 2 formulates this delivery problem as follows. Given the scalable media bit stream and the channel feedback (such as packet lost, round-trip-time), find the *best order* of packet delivery in the transmission buffer containing both the new packets and lost packets requesting for retransmission, in order to maximize the playback quality at the receiver end. This problem is solved by the proposed *scheduling* algorithm. A brief overview can be found in the end of this chapter (Section-1.5), and details are presented in Chapter 2.

1.3.4 Delivery over QoS networks

Compared to best-effort networks, QoS networks are more desirable for streaming applications, since the requirements, such as bandwidth, delay and packet loss rate, can be easily obtained in QoS networks. As a result, significant research work has

focused on (i) maximizing the video playback quality given certain QoS resources, and (ii) minimizing the resource usage for given streaming objects so that the system throughput can be increased, or the cost for the resource can be reduced.

Some of the joint approaches for best-effort networks described in the previous sections can also be applied in QoS networks. For example, JCSS is used in [33] for video transmission over the ATM networks with some policing constraints to maximize the video playback quality. In [78], a smoothing algorithm can be used to reduce the bandwidth that has to be reserved on the channel for VBR video transmission.

Chapter 3 (also see next section) includes an example of a proxy caching strategy for video delivery over QoS networks, which improves the channel bandwidth utilization (a video caching problem for best-effort networks is also covered at the same time).

1.4 Other Components in streaming video services system

1.4.1 Proxy caching for streaming video

The great majority of recent proxy caching research and development has focused on techniques that can handle generic web objects, i.e., a decision is made about whether an object should be cached based on the type of object, or on meta-data provided by

the content creator. But among "cacheable" objects there is no distinction is made between, say, an HTML text file and a JPEG image. Some recent work has proposed that having caching strategies that are specific of particular types of objects can help improving the caching performance. We refer to these approaches as "partial caching" methods to distinguish them from the "complete caching" for web objects. An example of partial caching for images is *soft caching* in [56, 98, 39], which results in images being "recorded" (i.e., compressed with lower quality), instead of simply being removed from the cache when there is not sufficient cache space left.

Likewise, caching only part of the video can be useful for the streaming video applications. For example, the *prefix caching* [84] caches the beginning frames of the video sequence in order to further smooth out the variance of the VBR video bitrate. Another approach in [101] caches part of larger frames (e.g., a larger frame is broken into two parts, one part is cached, the other is still in the server), in order to reduce the bandwidth requirement on the channel between server and proxy.

Chapter 3 proposes two streaming video proxy caching frameworks for the QoS networks and best-effort networks. Both of these two cases focus on the problem of improving the overall streaming and caching performance by using "partial" caching strategies, which cache only part of video on the proxy. In the case of QoS networks, the channels have high bandwidth, low loss rate and small delay, and the caching goal is to maximally reduce the bandwidth reservation needs (so that the network cost can be reduced), by selecting proper frames of the video to be cached. In the

case of best-effort networks, since the channels are vulnerable to congestion, delay and loss, the caching goal here is to improve the *robustness* of continuous playback at receiver end against bad network condition.

1.4.2 Disk storage strategy for central video server

For the video servers in VOD (video-on-demand) systems, the disk storage strategies (disk data placement) of video objects can be one of the critical factors that affects the server performance, i.e., the server *maximum throughput*, which can be measured by the maximum number of concurrent users can be supported by the server.

The factors that may affect the server throughput can be channel bandwidth, disk bandwidth and disk seek time. VOD servers are usually equipped with high speed disks with large capacities for video storage. Since the bandwidth of channel and disk are "hard" constraints, research has aimed at reducing the disk seek time, which is proportional to the distance the disk head has to travel to find the requested video data block. The studies in [30, 7, 9, 10] showed that by carefully designing the disk placement for the video objects, the disk seeking time for the video data blocks can be reduced, such that the server throughput is increased. In those schemes, the data blocks of different video objects are typically placed in a "sequential" fashion rather than being randomly placed. This is because that the video data is usually requested sequentially for playback, and a sequential placement can reduce the seek time significantly when the video size is very large.

For multiple users connected to the same server, a *Round Robin Service* [30] is applied to allocate disk bandwidth to all users, such that each user can receive a block of video data during a fixed interval in each service round. This block of data should be sufficient for the user to display video until the next block arrives. Most disk placement methods segment the video into fixed-size blocks and place them in a certain order so as to reduce the disk seek time for these blocks. Given that the video data is VBR in nature, the blocks with same size may not contain same number of frames to supply same playback duration.

The sequential placement implies restrictions in the access order of video blocks, which has a negative side effect: if a video block contains *fewer* frames to playback for a complete round for a particular user, the server has to (i) either ignore this and continue to serve other users, which will cause playback jitter for that particular user, or (ii) send additional blocks of data to that user, which requires the server to use additional disk seek time to retrieve other blocks, and may disorder the "sequential" data retrieval for all other users. In any case, the benefit of sequential data placement may become unavailable as the "random" disk seek has to be performed.

Since none of these two methods are desirable, Chapter 4 describes an approach to solve this problem in the compression domain, which compresses the video under the constraints of the particular disk placement so that each data block is guaranteed to have sufficient frames for playback during an entire service round, while the video quality is maximized.

1.5 Contributions of this thesis

There are three topics related to streaming media presented in this thesis, i.e., delivery, caching and disk storage.

1. Chapter 2 studies the end-to-end delivery strategies for scalable streaming media over best-effort networks. The starting point of this problem is how to make a decision on whether or not to retransmit a lost packet in the presence of delay constraints and limited channel bandwidth. It turns out that the delivery order of *all* packets (both the new and retransmitted packets) is important for the playback quality. This chapter proposes a fast scheduling algorithm to deliver the packets in sender's buffer, based on a packet importance metric that considers the data contents, data dependencies, channel conditions, and packet delivery status. The proposed algorithm can improve the streaming video (or audio) playback quality by 2 to 4dB, with very small computational overhead that enables it to be applied in real-time applications.

2. Chapter 3 focuses on the video caching problem for both QoS networks and best-effort networks. This chapter shows how to cache only *part* of the video to improve the overall performance when proxy cache space is limited. For QoS networks, the caching goal is to reduce the network bandwidth cost and channel utilization. The proposed method is a frame-based selection algorithm, which uses an iterative method to select frames to be cached, such that the channel bandwidth is guaranteed to be (maximally) reduced.

For best-effort networks, the caching goal is to improve the robustness of continuous playback against poor network condition, Another frame-based iterative caching algorithm is proposed for this case, where a frame is selected to maximally improve the robustness. Both caching algorithms take the client buffer size into account so that the caching performance is maximized without producing overflow in the client buffer.

3. Chapter 4 formulates a rate-distortion optimal video compression problem under the constraints of particular disk storage strategies for video servers. It proposes an R-D based compression algorithm such that the video quality is maximized under the condition that continuous playback is guaranteed even when the server throughput reaches its maximum limit. This chapter transforms the disk placement constraint into video rate constraints, and applies the Multiple Lagrange Multiplier during the video compression stage to determine the optimal (in an R-D sense) operating points for compressed video.

Chapter 2

Scalable streaming media delivery

2.1 Introduction

In this chapter we focus on the problem of delivery of scalable streaming media over a lossy packet network, e.g., the Internet. More specifically, we consider an end-to-end system, where a video server and client are connected through a channel, suffering from packet loss and varying channel bandwidth. Our goal is to find a packet transmission policy to select the packets to be transmitted (or retransmitted) at any given time during a streaming session, in such a way as to maximize the playback quality. In this Chapter we propose a server-driven "packet scheduling" algorithm especially designed for scalable media.

Before we introduce the main ideas of this chapter, we first give some examples of media delivery over the best-effort networks. First, consider a video sequence encoded in a single layer and without any temporal dependencies (e.g, motion JPEG). To transmit such a video object can be straightforward: we can simply transmit the

frames according to their original their playback time, i.e., frames to be displayed earlier will be transmitted earlier (due to the delay constraint). A second example is to transmit video encoded with bi-directional temporal dependencies (e.g., MPEG encoded video). For example, when B-frames are used, it may be necessary to transmit all the reference frames before transmitting any of the B-frames. This is because B-frames are useless unless their reference frames have arrived.

The last example is for a scalable video encoded with both temporal and SNR (or other) dependencies, where each frame may contain several layers, e.g., the Fine Grain Scalability (FGS) algorithms with MPEG 4 [3]. The base layer can be decoded by itself with low quality; while the higher layer can be decoded, with the presence of the entire base layer, to obtain enhanced visual quality. The data dependencies among both layers and frames introduce a more complex dependency relationship across the different video data packets.

Furthermore, when streaming over a best-effort network with packet loss or error, retransmission can be used to recover the lost packets, as long as the delay constraints are met. However a retransmitted packet typically has an extra delay of one or more RTT(s) (round-trip-time), and can not be guaranteed to arrive to the client on time. In addition, even if there is still time to retransmit a given packet, a decision needs to be made on whether this packet should be given more preference over other retransmission requests.

In this Chapter, we propose a scheduling algorithm to select the most "important" packets to be transmitted at a given time, i.e., the packets that will tend to maximize the playback quality, where "importance value" of a packet is evaluated by taking into account data dependencies, network conditions and other factors. The work in this Chapter is an extension of our previous work [51] where we apply the scheduling algorithm to layered video objects with more complex dependencies than those considered in [51]. Our simulation results show that by using the proposed algorithm, the video playback quality (in PSNR) can be improved around 2 dB, compared to a simple fixed scheduling approach, the when packet loss rate is around 20%.

Retransmission with delay constraints has been studied in [60, 42], where the decisions have been made on whether to retransmit or not based on the time-out factor the lost packet. While those works consider the problem of retransmissions, they do not consider specifically scalable media, which is addressed in our work.

In [65] a rate control algorithm for delivering MPEG-4 video over the Internet was proposed with a priority re-transmission strategy for recovery of lost packets, which incorporates a the constraint to prevent the receiver buffer underflow. This is achieved by giving priority to retransmission of base (lower) layers. However this work did not address the problem of the delivery order of new packets in the sender's buffer. For example, the enhancement layer is packetized before transmission, and those packets actually have decoding dependencies on each other as well as the

dependencies on the base layer. Our proposed framework solves the problem of delivery order of both the new and lost packets and can complement other rate-control based schemes, which make decisions on the rate of the video bit-stream to be transmitted, e.g., [65].

Optimization of layered streaming media delivery was also addressed by Podolsky *et al* [64], who use a Markov chain analysis to find the optimal packets transmission and retransmission policies. However, the algorithm in [64] needs to search over all the possible candidate policies and thus the policy space grows exponentially with the number of layers and frames (in the scheduling window). In practice this algorithm may only be used with a limited number of layers and frames. Chou and Miao also addressed the same problem with a rate-distortion analysis [15, 16]. They show that the policy space can be factored so that packet dependencies are loosely coupled, and the optimal schedule policy can be found through a few of iterations over all the packets in the scheduling window. Therefore that framework can operate on more layers and a larger transmission window.

The main contribution of our work is to propose a simplified formulation and cost function so that a decision on which packet to transmit can be found without any iterations, reducing the complexity greatly with respect to [15, 16]. This is achieved by introducing the concept of expected run-time distortion, which summarizes the parameters to be considered in the packet scheduling (such as data dependencies, network condition) into a single packet importance metric. A similar concept of

packet expected distortion has been proposed in [18, 19] and also used in [15, 16]. Those metrics take into account the status of packets whose decoding is needed before the current packet can be decoded (i.e., its parent packets). Our definition of expected run-time distortion extends this metric by explicitly including the "importance" of the children packets (i.e., those that depend on the current one). While the iterative techniques in [15, 16] resolve the data dependencies and take into account the impact of future scheduling of other packets, we can achieve these without iterations by using our proposed metric.

Therefore our proposed scheduling scheme can operate, with very low computational overhead, on more complex media streaming, such as MPEG 4 FGS scalable video, where a video packet may have hundreds of parent or children packets in the dependency tree (see details below). Furthermore, our proposed low complexity algorithm is no longer restricted to operate on fixed transmission times as in [64, 15, 16] (i.e., the scheduling algorithm assume packets sent at fixed intervals). Our packet selection algorithm can be used at any desired time when policy is needed to select packet(s) for transmission (or retransmission). Thus data packet size can be either fixed or variable.

The chapter is organized as follows. In Section 2.2 we describe the backgrounds for scalable media (e.g., MPEG 4 FGS) data, the data packetization and the streaming system architecture used in this chapter. Section 2.3 formulates our packet

scheduling problem. Section 2.4 presents the proposed ERDBS algorithm, and simulation results is shown in Section 2.5. Section 2.6 concludes the chapter.

2.2 Scalable Media and Streaming System Architecture

Scalable encoded media has several layers. The lowest layer (or base layer) can be decoded by itself to obtain a coarse reconstructed version of the signal. As increasing numbers of higher layers to be decoded, more refined version of the signal is achieved. The MPEG 4 standard specifies a FGS (Fine Grain Scalability) mode [3], where a video sequence can be encoded into two layers: a base layer and an enhancement layer. The base layer has to be received completely before it can be decoded; while to obtain better reconstruction quality, one can continue to decode the enhancement layer, and can stop decoding at any position of the enhancement layer.

2.2.1 Data packetization

In this chapter we use the MPEG 4 FGS video stream as implemented by the reference codec [3]. The base and enhancement layers are packetized separately, see Fig. 2.1(a). Usually the enhancement layer has large size and in order to generate different layers out of a single FGS layer, the enhancement layer is packetized into several packets; while the base layers of many frames can be packetized into a single packet.

For each packet, denoted as $l_{i,j}$ (the j^{th} layer of the i^{th} frame), we are interested in the following parameters: (i) the size of the packet $r_{i,j}$; (ii) the playback time of the packet t_i, which is defined as the time when frame i has to be displayed at the client end; (iii) the distortion value of the packet $d_{i,j}$; (iv) the set of its parent packets $\mathcal{A}_{i,j}$; and (v) the set of its children packets $\mathcal{B}_{i,j}$.

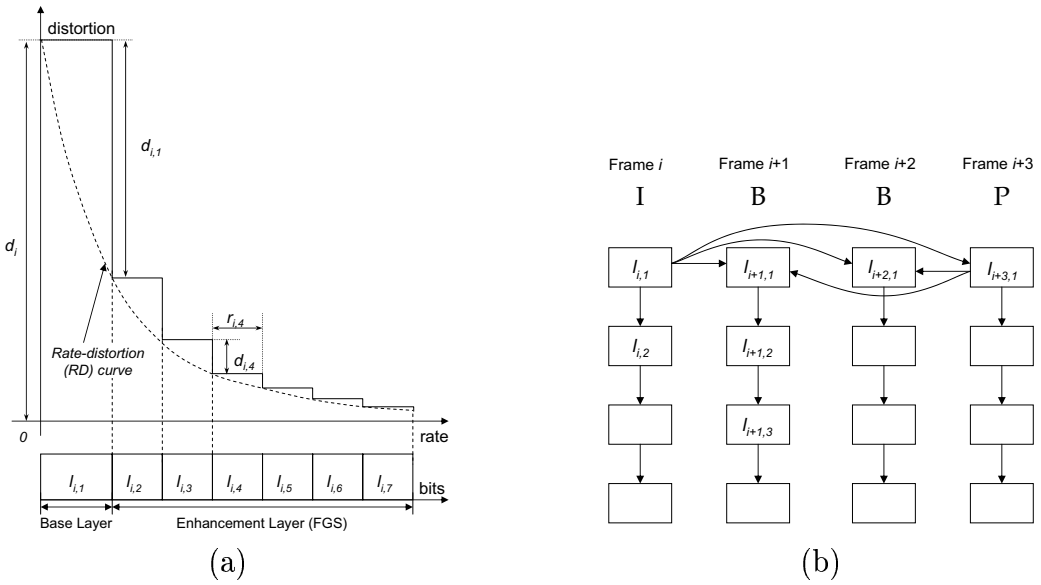

Figure 2.1: (a) MPEG 4 FGS packetization of a single frame. (b) MPEG 4 FGS packetization of frames with inter-frame dependencies.

The packet distortion value $d_{i,j}$ is determined as the decrease in the distortion of the reconstructed signal when the corresponding packet $l_{i,j}$ is decoded. For example in Fig. 2.1(a), when packet $l_{i,4}$ is decoded, the distortion is reduced by $d_{i,4}$. If the base layer is split into several packets then the distortion value of each base layer packet can be assigned proportional to its packet size $r_{i,j}$, while their sum is equal to the total distortion value of the base layer. However it should be emphasized that

this distortion value assignment is for our scheduling algorithm proposed below, decoding any single packet of the base layer cannot decrease the distortion of the reconstructed signal, as the base layer (of a frame) cannot be decoded without all its packets being available.

The parent set $\mathcal{A}_{i,j}$ of packet $l_{i,j}$ is defined as the set of packets that has to be available (or decoded) in order to decode packet $l_{i,j}$. For example $\mathcal{A}_{i,4}$, the parent set of packet $l_{i,4}$, includes packets $l_{i,1}$, $l_{i,2}$ and $l_{i,3}$. The child set $\mathcal{B}_{i,j}$ of packet $l_{i,j}$ is defined as the set of packets that cannot be decoded without the presence of packet $l_{i,j}$, i.e., $\mathcal{B}_{i,j}$ contains all packets that have packet $l_{i,j}$ in their parent set. For example, the child set of $l_{i,4}$ is $\mathcal{B}_{i,4}$ including packets $l_{i,5}$, $l_{i,6}$ and $l_{i,7}$.

Fig. 2.1(a) is an example for a single frame with only intra-frame dependencies (e.g., an I frame). For any P or B frame, the inter-frame dependencies should be taken into account to form the parent and child sets, which may contain packets across several frames. For example, in Fig. 2.1(b), when two packets are connected by a line, this means there is a direct dependency between them. The arrows are directed from parent to child packets. Note that some parent packets of a packet $l_{i,j}$ may not be shown explicitly, but should be counted as well. For example, packet $l_{i+1,2}$ has a direct parent $l_{i+1,1}$ and other parent packets $l_{i,1}$ and $l_{i+3,1}$ (this is because packets $l_{i,1}$ and $l_{i+3,1}$ are in the parent set of packet $l_{i+1,1}$).

Although the original MPEG 4 FGS stream has two layers, after packetization, the enhancement layer can be broken into several sub-layers, where each packet

represents a sub-layer. We define the total distortion of a frame, d_i, as the distortion when a frame is completely lost (see Fig. 2.1(a)). A frame reconstructed with all its layers (packets) will have the minimum distortion. Define the total distortion of the media stream as the sum of d_i of all the individual frames. The playback distortion is defined as the total distortion minus the distortion of all layers and frames that are actually used at playback. Define $a_{i,j}$ as an indicator such that $a_{i,j} = 1$ if $l_{i,j}$ is *used* for playback, otherwise $a_{i,j} = 0$. For a media sequence with N frames, with L_i layers in each frame, the *actual playback distortion*, D_V, can be obtained as

$$D_V = \sum_{i=1}^{N} \sum_{j=1}^{L_i} d_{i,j} - \sum_{i=1}^{N} \sum_{j=1}^{L_i} a_{i,j} d_{i,j}. \tag{2.1}$$

Note that the on-time arrival of a packet $l_{i,j}$ does not necessary mean that it can be used for playback, as decoding a packet is not possible unless all its parent packets have also arrived on time.

2.2.2 System architecture

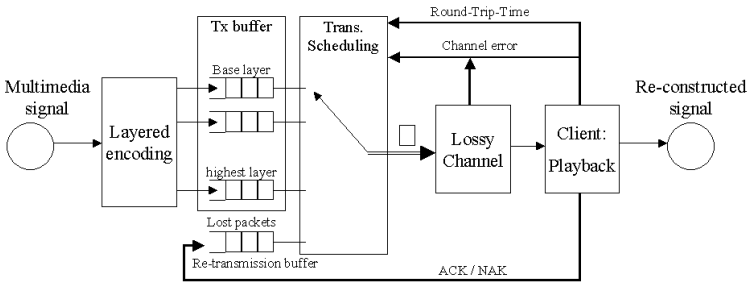

Figure 2.2: System architecture

Fig. 2.2 shows the architecture of our proposed streaming system. The server and the client are connected by an unreliable channel link, which has low delay but with packet loss/delay (e.g., a UDP channel). We denote ϵ the packet loss probability in this server-client data channel. The source media sequence is compressed and packetized into several layers/packets, then is fed into the server's transmission buffer. These packets are referred as the "new" packets waiting to be scheduled for transmission. There are also packets in the server's buffer that are waiting for retransmission because they are reported lost or no acknowledgment was received. The server's scheduling module selects one packet at a transmission time from those buffers and sends it to the channel. At the client end, the lost or damaged packets are reported to the server via a feedback channel.

The value of round-trip-time (RTT) is defined as the interval from the time a packet is sent from the server to the time the server gets feedback on this packet from the client. With a smaller RTT, the server can get the feedback more promptly and have more time to re-send (if necessary) a packet before its time-out. If the feedback channel is also unreliable, RTT estimates can be used by the sender as the "time-out" threshold in the case of lost of feedback. Both channel error and RTT can be estimated by the feedback information. They can be used by the server to adjust the stream delivery mechanism according to the varying channel conditions.

The start-up delay, τ, is defined as the period between the time when the first packet is sent by the server, and the time when the first frame starts to be displayed

at the client. Usually a larger initial delay can smooth out more variations in channel behavior, and enable more time for retransmissions, resulting in a better playback quality. Refer to [32, 50, 49] for a more detailed analysis of the trade-off between start-up delay and playback quality. Our proposed scheduling framework can be applied for both pre-recorded and real-time encoded media which has a small start-up delay.

2.3 The Packet Scheduling Problem

The goal of this work is to minimize the playback distortion D_V for a streaming session in a lossy packet network. Due to the fixed start-up delay constraint, not all lost packets can be recovered by retransmission. However, if the server schedules a packet to be sent much earlier than its playback time, this packet will have more chances to be retransmitted (and received) before it is too late for display. This fact motivates the main ideas in our proposed framework.

Given a set of packets, $\mathcal{G}$, that are the candidates to be transmitted by the server, we define a schedule s as the transmission order of all those candidate packets, which specifies whether and when a packet should be transmitted. Clearly, the order of delivering the packets has an impact on the actual playback quality, because of the delay constraint (which limits the retransmission) and data dependencies. Intuitively, it would be useful to transmit (retransmit) a more "important" packet at an earlier time, e.g., a base layer or I frame packets, rather than simply transmit

all the packets in a sequential order (i.e., following the original time stamp order) without any distinction among all the packets.

Due to the packet loss, the importance of a packet needs to take into account not only the playback distortion $d_{i,j}$ and its playback time t_i, but also the status of other packets in both the parent and child sets. These other parameters can be captured into an "expected" distortion metric. We denote $\hat{D}_G$ as the *expected playback distortion* of the packet set $\mathcal{G}$, where $\mathcal{G}$ is the set of packets currently available in the transmission buffer. Similar to (2.1), we have

$$\hat{D}_G(s) = \sum_{l_{i,j} \in \mathcal{G}} d_{i,j}, - \sum_{l_{i,j} \in \mathcal{G}} \hat{a}_{i,j} d_{i,j}, \tag{2.2}$$

where $\hat{a}_{i,j}$ is the probability that packet $l_{i,j}$ *can* be used for decoding,

$$\hat{a}_{i,j} = \prod_{l_{m,n} \in \mathcal{A}_{i,j}} (1 - p_{m,n}(s)). \tag{2.3}$$

Eq. (2.3) specifies that the probability, $\hat{a}_{i,j}$, is the product of the probabilities of the successful arrival of all its parent packets. $p_{m,n}(s)$ is the probability that packet $l_{m,n}$ (which may or may not belong to $\mathcal{G}$) is lost or delayed, for a given schedule s. Clearly $p_{m,n}$ depends on both the packet loss rate ϵ and the schedule s. For a given ϵ, if packet $l_{m,n}$ is scheduled to be transmitted earlier, it will have more chances to arrive on time, or to be retransmitted (within the delay constraint) if it is reported lost. We can formulate the scheduling problem as follows.

Given a set of packets $\mathcal{G}$, and given τ, ϵ, RTT, find the optimal schedule $s^* \in \mathcal{S}$, such that the expected playback distortion $\hat{D}_G$ is minimized, when all packets in $\mathcal{G}$ are transmitted at the time specified by s^*,

$$s^* = \mathbf{arg} \min_{s \in \mathcal{S}} \hat{D}_G(s), \tag{2.4}$$

where $\mathcal{S}$ is the set containing all possible schedules. Note that after the first packet in s^* is sent, the set $\mathcal{G}$ will change accordingly because the transmitted packets is removed and possibly other packets are added to the set. Therefore s^* has to be re-computed after each packet is sent during the streaming session. In other words, the server is only interested in the *first* packet in s^* at any given time that a packet need to be selected for transmission. This property will be used in our proposed algorithm below.

2.4 Proposed Scheduling Algorithm

The set of candidate packets $\mathcal{G}$ can contain packets of several frames, or the entire video sequence. For a streaming session where the source is captured in real-time, it is limited by the end-to-end delay τ. In general, we can use a "look-ahead" sliding window to specify which of the new packets should be considered as part of the set $\mathcal{G}$. Note that the candidate packet set $\mathcal{G}$ also includes the packets that are reported as lost and waiting for retransmission.

To find the optimal solution for our problem in (2.4), an exhaustive search over all candidate schedules is possible but may not be practically useful, since even a small candidate set $\mathcal{G}$ can leads to a large number of candidate schedules. The size of $\mathcal{S}$, and therefore the search complexity, increases exponentially with the size of $\mathcal{G}$. Exhaustive search is only applicable when the window size W is very small. Alternatives to an exhaustive search do exist as in the search algorithms proposed by Chou and Miao in [15, 16].

In this chapter we propose a fast heuristic approach to solve this problem practically. The performance of the heuristic approach is very close to that of an exhaustive search (see experimental results below). At the end of this section we will discuss the difference between our approach and those in [15, 16].

2.4.1 Expected run-time packet distortion

Recall that only the first packet in s^* is actually used for transmission at a given time. Thus, instead of scheduling the order of the transmission of a packets in $\mathcal{G}$, if one can properly predict the importance of each packet in $\mathcal{G}$, then choosing the most "important" packet to be (re-)transmitted should provide a greedy solution to the scheduling problem.

In order to estimate the "importance" of a packet prior to the transmission, we propose the concept of *expected run-time distortion*, which takes into account (i) the packet distortion $d_{i,j}$, (ii) the packet data dependencies, (iii) the channel conditions

(e.g., loss rate, RTT), (iv) the packet playback time (deadline) and its delivery status (e.g., transmitted, received, lost, etc.).

We first introduce the concept of *run-time distortion*, $\hat{d}_{i,j}$, which enables us to capture the dependencies among packets (layers). We will show some examples before giving its definition. Consider transmitting or retransmitting a packet $l_{i,j}$. If it is *independent* from any other packets, its run-time distortion is equal to its original distortion, $\hat{d}_{i,j} = d_{i,j}$. If it has a child packet $l_{i,j+1}$, we will have the following situations.

1. If $l_{i,j+1}$ has not been transmitted yet, transmitting $l_{i,j}$ only affects the layer itself.

2. If $l_{i,j+1}$ has been transmitted (but without any ACK or NAK), transmitting $l_{i,j}$ becomes more valuable since $l_{i,j+1}$ might be received and has to be decoded with $l_{i,j}$.

3. If $l_{i,j+1}$ has been transmitted and an ACK feedback is received, then $l_{i,j}$ becomes more important, because the received $l_{i,j+1}$ will be useless without this $l_{i,j}$.

4. If $l_{i,j+1}$ has been transmitted and a NAK was received we will have the same situation as in (1).

Similarly, the gain of transmitting (re-transmitting) $l_{i,j+1}$ also depends on the status of $l_{i,j}$. Extending the above to multiple layers, we define the run-time distortion of $l_{i,j}$ as follows:

$$\hat{d}_{i,j} = d_{i,j} \prod_{l_{m,n} \in \mathcal{A}_{i,j}} (1 - P^H(l_{m,n})) + \sum_{l_{m,n} \in \mathcal{B}_{i,j}} d_{m,n}(1 - P^H(l_{m,n})), \tag{2.5}$$

where $P^H(l_{m,n})$ is probability of loss/damage to layer $l_{m,n}$, based on its transmission history, and is defined as

$$P^H(l_{m,n}) = \begin{cases} 1 & \text{if layer } l \text{ has not been sent,} \\ 1 & \text{if there is NAK on layer } l, \\ \epsilon^n & \text{if layer } l \text{ has been sent } n \text{ times,} \\ 0 & \text{if layer } l \text{ is ACKed.} \end{cases} \tag{2.6}$$

The term $d_{i,j} \prod_{l_{m,n} \in \mathcal{A}_{i,j}} (1 - P^H(l_{m,n}))$ in (2.5) shows that the original distortion of a packet is weighted by the probability of receiving all its parent packets. The second term, $\sum_{l_{m,n} \in \mathcal{B}_{i,j}} d_{m,n}(1 - P^H(l_{m,n}))$, indicates that the importance of a packet increases if any of its children packets has been received. Before anything is transmitted, the run-time distortion of all layers is zero except for the lowest layer, for which the run-time distortion is equal to the original distortion $\hat{d}_{i,j} = d_{i,j}$. Eq. (2.5) implies that only after transmission (at least once) of all its parents, does a packet's run-time distortion become non-zero (except the base layer). The run-time distortion increases if a child packet has arrived (or has been transmitted). Thus this definition

reflects the "importance" of a layer given data distortion, data dependencies, and the current status of the transmission.

Now we consider the loss probability of packet $l_{i,j}$ when we are able to send it at time $t^X_{i,j}$. We denote it as $P^X(l_{i,j})$ to distinguish it from P^H in (2.6). Since there could be possible retransmissions for a packet $l_{i,j}$ before its playback time t_i, we approximate $P^X(l_{i,j})$ as follows,

$$P^X(l_{i,j}) = \epsilon^{u_{i,j}}, \tag{2.7}$$

where $u_{i,j}$ is the number of possible retransmissions before packet $l_{i,j}$ passes its playback deadline, and can be obtained as

$$u_{i,j} = (t_i - t^X_{i,j} - t^D_{i,j})/RTT, \tag{2.8}$$

where $t^D_{i,j}$ is the transmission delay for $l_{i,j}$, i.e., $t^D_{i,j} = r_{i,j}/C(t)$. $C(t)$ is channel bandwidth, which can be varying over the time.

We here defined the *expected run-time distortion* $\widetilde{d_{i,j}}$ of a packet $l_{i,j}$ as follows,

$$\widetilde{d_{i,j}}(t^X_{i,j}) = P^X(l_{i,j}) \times \hat{d}_{i,j}, \tag{2.9}$$

where $\widetilde{d_{i,j}}$ is a function of $t^X_{i,j}$ since $P^X(l_{i,j})$ depends on it. We will use $\widetilde{d_{i,j}}$ as the importance metric of packet $l_{i,j}$.

2.4.2 Expected Run-time Distortion Based Scheduling – ERDBS

We now propose a fast scheduling algorithm to select packets for transmission at any given time. Denote t_{cur} as the current time at which the server wishes to select a packet to transmit. Note that t_{cur} corresponds to the time when the first packet in a schedule s is to be sent. We substitute $t_{i,j}^X$ in (2.9) by t_{cur}, for all the packets in $\mathcal{G}$, and select the packet with the largest value of $\widetilde{d_{i,j}}(t_{cur})$ to be transmitted at current time t_{cur}. The reason is that the expected run-time distortion evaluated at the current time t_{cur} reflects importance of the packet *if* it is to be transmitted at t_{cur}.

We refer to this approach as "Expected Run-time Distortion Based Scheduling" (ERDBS). The complexity is greatly reduced with this algorithm, since only one iteration for all the packets in transmission buffer is needed. The results in [51] show that the packets selected using this greedy search, are almost identical to the ones selected by the optimal schedule in (2.4) using an exhaustive search.

2.4.3 Discussion

From (2.2) and (2.3), we can see that the distortion decrease in $\hat{D}_G(s)$ achieved by any given single packet $l_{i,j}$ is heavily coupled with the transmission schedule of its parent packets. This introduces high complexity in the search for the optimal scheduling solution. As mentioned before, Chou and Miao [15, 16] already show that the dependencies of packets can be factored in a way such that the optimal

schedule policy can be found with several iterations instead of a full search. The main difference between the approach in this chapter and those in [15, 16] lies in the following two aspects. First, for any packet $l_{i,j}$, the status and distortion values of its child packets have been explicitly taken into account in our definition of the run-time distortion (see (2.5)). Second, the loss probabilities of all parent and child packets of $l_{i,j}$ are computed based *only* on the history of the transmission of those packets (see (2.6)), without any assumption of the future transmission schedules. In contrast, in [15, 16] these probabilities are calculated based on both the history and the transmission policies which are scheduled in the future.

Therefore, with (2.5) and (2.7) we are able to capture both the data dependencies and delay constraints into a single importance metric, i.e., *expected run-time distortion*, and the selection of packets can be done by choosing the packet with that maximum metric. Thus the full search in [64] or the iterative techniques in [15, 16] can be avoided.

2.5 Experimental Results

To evaluate our proposed scheduling algorithm, we also simulate a transmission scheme with no particular scheduling among different packets. We refer to this as the *Sequential Scheduling (SS)* scheme. In SS, the data packetization is the same as depicted in Section 2.2.1. The packets within the same frame will be transmitted consecutively. The frames are delivered according to their original order in the

encoded video stream. The SS scheme has the ability to discard a packet (either new or retransmitted) if it detects that this packet exceeds its playback time, in order to save bandwidth for other packets. The SS scheme selects proper layers of each frame to be sent to the client such that the average video data rate will not exceed the available channel bandwidth, and the remaining higher layers will be discarded.

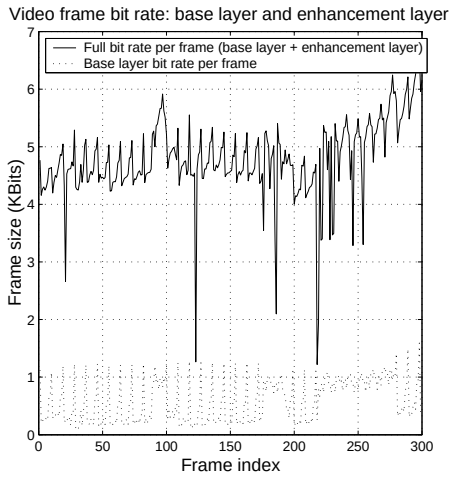

Figure 2.3: MPEG 4 FGS frame data rate. The dotted line is the base layer size of each frame; and the solid line represents the full size of each frame (base layer and enhancement layer).

We use a MPEG 4 FGS video stream in our simulation. The video data is packetized with a fixed packet size of 512 bytes. The data used in the simulation is shown in Fig. 2.5 (we use the video sequence *Stefan* with 300 frames). The average video data rate after compression, with both base and enhancement layer, is 958 KBits/second. The channel packet loss rate is 0.2, i.e., 20% of the packets will be lost when transmitting over the server-client data channel, and packet loss is assumed to be i.i.d.. The round-trip-time is 200 ms and the streaming session has a start-up

delay of 40 ms. The playback quality is measured by PSNR of the reconstructed video frames at the client end incorporating all the packets that were received on time. Over 100 realizations are averaged in the results we present.

Fig. 2.4 shows the simulation results of the playback quality using the FGS video stream (in Fig. 2.5) with various parameters (such as bandwidth, channel packet loss rate, start-up delay and RTT). Our simulation shows that by using the proposed ERDBS algorithm to deliver the video data, the playback quality improves about 2 dB compared to using a SS delivery algorithm.

2.6 Conclusions

In this chapter, a new framework for delivery of scalable streaming media data over networks with packet loss is presented. We proposed a new delivery method, ERDBS, for this framework to solve the packet scheduling problem. The proposed algorithm, designed for a sender-driven transmission system, can increase the client playback quality by selecting proper packet(s) to be transmitted at any given time during the streaming session. The simulation results shows that ERDBS algorithms outperforms the other packet delivery schemes with a fixed scheduling, in the presence of channel packet loss. The low complexity of the search algorithm in ERDBS also enable it to be applied in real-time applications.

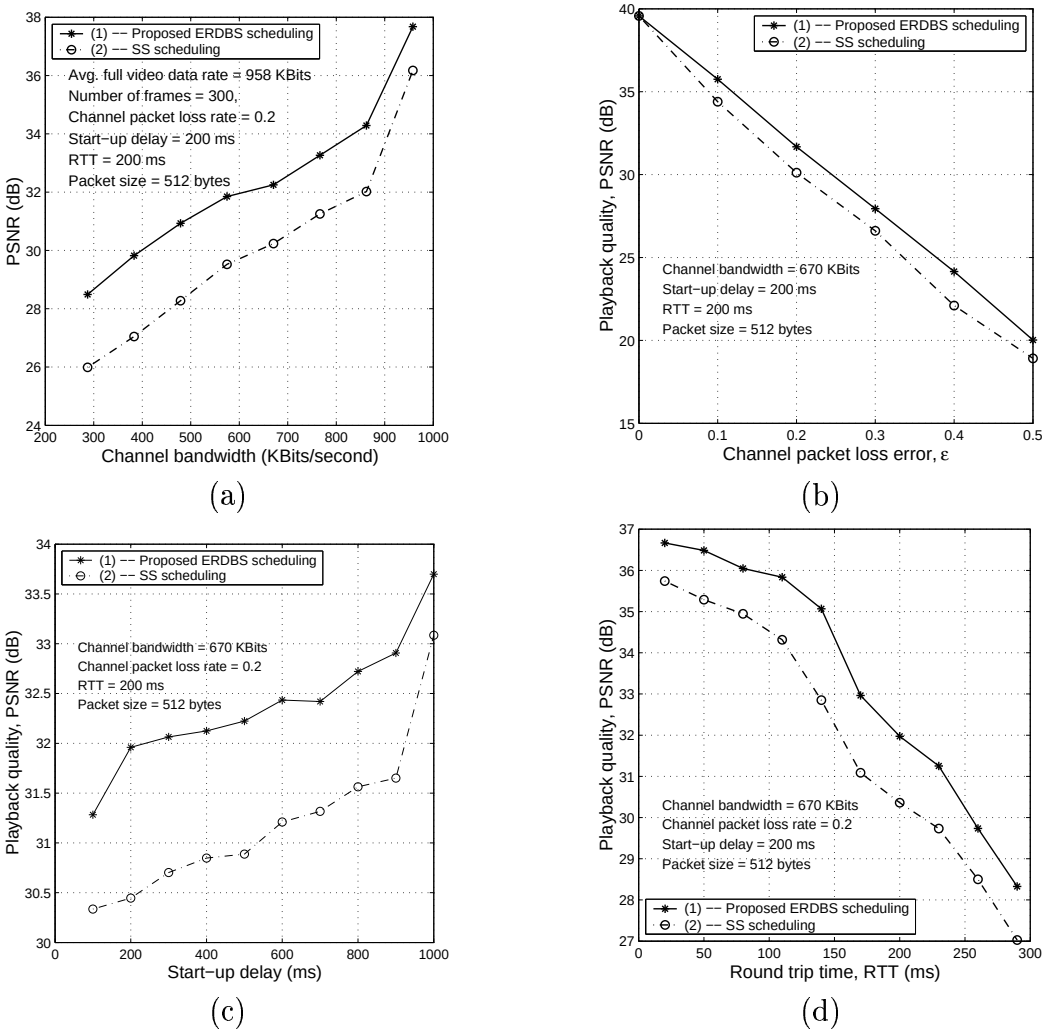

Figure 2.4: The comparison of playback quality (PSNR) using proposed ERDBS and SS algorithm, in various conditions, such as bandwidth, channel packet loss rate, start-up delay and round trip time. In most cases the playback quality of ERDBS outperforms the regular SS delivery algorithm around 2 dB.

Chapter 3

Scalable Proxy Caching for Streaming Video Applications

3.1 Introduction

Interest in proxy based caching has increased with the growth in Internet traffic and the initial research in this area (e.g., within the Harvest project [11]) has quickly led to the development of commercial products and continuing research activity (e.g., [96, 87, 88, 75, 44, 94]). The great majority of recent research and development on proxy caching has focused on techniques that can handle generic web objects; among the "cacheable" objects no distinction is made between, say, an HTML text file and a JPEG image. As real-time streaming video is becoming a significant proportion of network traffic and, given the large data volumes involved and its variable-bit-rate (VBR) nature, even a few popular video applications can result in potential network congestion problems. The congestion can cause not only packet loss, but also packet

delays, which degrade the video playback quality dramatically (since the packets that arrive after their playback time are useless).

In this chapter we focus on the caching problem specifically for streaming video objects. Some recent work has proposed that having caching strategies which are specific for particular types of objects can help improving the overall performance. In particular, for some objects, it is possible to perform "partial caching", where only part of the objects are stored on the proxy, as opposed to the "complete caching" where the objects are stored completely. An example can be found in "soft caching" for images ([56, 98, 39]). Approaches for partial caching for video have also been proposed in [101, 84]. In this chapter, we study a "selective caching" strategy [50], which selects only *certain parts* of the video to be cached. We will show that the strategy to use depends on the specific network environment; and we focus on two representative scenarios, namely networks with Quality-of-Service (QoS) and best-effort networks. We provide a selective caching method for each of these scenarios.

Our proposed caching methods are *frame-wise* selection algorithms, i.e., the smallest caching unit we consider is one frame of the video (each frame may contain different number of bytes). Since there can be frequent changes on caching parameters (e.g., the popularity of the video objects), it is desirable to enable the proxy to be *scalable*, i.e., to be able to easily increase/reduce the portion of video being cached while still maintaining good performance. This scalability is inherent in

our proposed frame-wise selective caching methods, by which the proxy can simply add/drop the selected frames as the environment changes.

3.1.1 System architecture

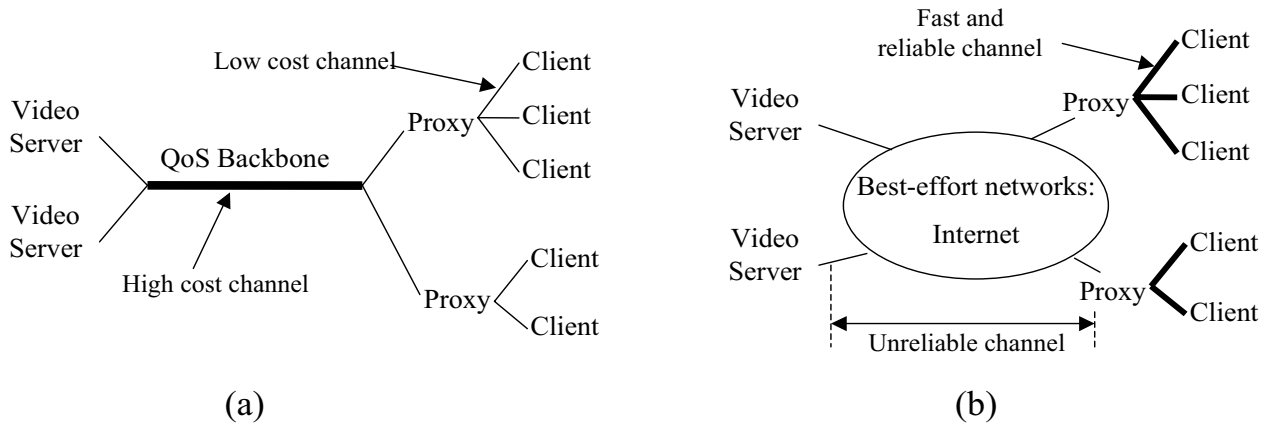

Figure 3.1: System architecture. The proxies are set close the clients, and are connected to the video server via either a QoS (a) or best-effort network (b).

The server-proxy-client model used in this chapter is shown in Fig. 3.1. The clients are attached to the proxy and all their requests for videos go through the proxy. The proxy allocates a certain cache space for each video sequence. Upon the client's request, if the frames are cached, the proxy will send them from its cache to the client directly; otherwise, the proxy will retrieve the frames from the video server. For a video streaming session, usually there is an end-to-end playback delay d, which is the interval between the time when the first packet is sent and the time when the first frame is displayed. For a continuous playback, the transmission of any given frame cannot exceed the delay d. We are also interested in other parameters for the design of selective caching, such as the server-proxy and proxy-client channel

characteristics; the cache space (H) allocated on the proxy for a particular video; and client buffer size (B_c). We consider two network cases in the following and will propose different strategies for selective caching for each of them.

Case 1: Proxy caching in QoS networks (see Fig. 3.1a). The bandwidth on the server-proxy channel can be reserved, and the cost of reservation is proportional to the reserved bandwidth. The goal of caching in this case is to reduce the amount of bandwidth C that has to be reserved on the server-proxy backbone channel (therefore reducing the network cost), while minimizing the required client buffer size B_c to achieve that reduction, given a limited cache space H.

Case 2: Proxy caching in best-effort networks without QoS on the server-proxy channel, as for example in the current Internet (see Fig. 3.1b). The delivery of data over these channels is vulnerable to congestion, delay and loss, which are harmful for real-time streaming video delivery. The caching goal for this case is to provide more robustness[1] for continuous playback against poor network conditions on the proxy-server channel.

To improve the streaming performance given that the proxy-client channel is fast and reliable, one could consider using the proxy as an external buffer for each client, i.e., such that a client with minimal buffer resources can store some of the incoming video data at the proxy. In this case the proxy would need to allocate separate storage resources to each client for each streaming session (even when some

[1]See Section 3.4 for detailed definition of robustness.

sessions may be accessing the same video object). Thus, as a proxy may serve a large number of clients simultaneously, this scenario may require the availability of high performance caching resources at the proxy, including not only memory but also output bandwidth. Both these resources have to increase as the number of clients is increased, since all the clients active at any given time will be simultaneously using the proxy for secondary storage.

Thus, in this chapter, we focus on a less resource-intensive caching scenario where caching storage is assigned to *each video object*, rather than to each client. The assumption here is that the data stored for each video object changes only when the popularity of the video object changes and that the storage is shared by all clients accessing a given video object. Therefore the requirement of the cache storage space is reduced. In additional, this approach reduces the amount of data that the proxy has to provide to the clients (since only certain video frames need to be served from the proxy) and also requires substantially less real time storage management (since only when the popularity of an object changes is the storage devoted to it modified.)

In both cases, the proxy-client channel delivers the data originated from both server and proxy. In our model we assume that the main network bottleneck (in terms of both cost and reliability) is the server-proxy channel, and therefore the proxy-client link is assumed to have "left-over" bandwidth even when video transmission is ongoing. One example of such a scenario would be accessing a relatively low bandwidth video stream through a DSL/cable link. Another example would

that where proxy and client are co-located within the same local area network [101]. The algorithms proposed here are good approximations for the case when there is a substantial amount of "left-over" bandwidth, in which we can assume that the delay in delivering a frame from the proxy to the client is very small (even when transmission of other frames from the server is taking place simultaneously.) This will enable us to assume that, since they can delivered in a very short time, frames stored at the proxy consume practically no client buffer memory. Therefore, to simplify the analysis, we exclude them from client buffer consumption in the rest of this chapter. For the scenario when when the "left-over" bandwidth on proxy-client channel is not large enough to ignore the delay between proxy and client, our buffer analysis can still be used as an approximation for the proposed selective caching algorithms. In the extreme case where there is no "left-over" bandwidth, and therefore the client cannot receive data simultaneously from the proxy and the server, our proposed will reduce to the prefix caching proposed in [84], which will provide the optimal solution.

We will show that the performance of some "partial" video caching strategies may be limited by client buffer size constraints. For example, it is obvious that in QoS networks caching any part of the video can reduce the server-proxy bandwidth C, since less data has to be retrieved from the server directly. However, the bandwidth may be reduced at the expense of increasing in the required client buffer size B_c, because the frames that are stored at the server (not cached at the proxy) will have to

be buffered at the client until they are displayed. We will show that the increment in B_c varies among different selective caching strategies which lead to the same bandwidth reduction. Similar constraints also exist in the case of video caching for best-effort networks. Therefore, there are trade-offs between the reduction of C (or improvement of U) and the memory size B_c to achieve it. Our goal is to *find proper selective caching methods such that the best performance is achieved under a constraint on* B_c. Thus one of the main differences between our work and other proposed caching algorithms (see below) is that we will consider the storage constraints at *both* client and proxy.

In summary, the main assumptions we make in our work, which also explain how it differs from other proposed caching algorithms (see below), are (i) that memory at the client is constrained, (ii) that cost considerations preclude using the proxy cache to provide dynamic "additional memory" for each client buffer, and (iii) that there is some "left-over" bandwidth in the proxy-client link. We will propose two approaches for selective caching in each of the above cases, namely, *Selective Caching for QoS networks* (SCQ) and *Selective Caching for Best-effort networks* (SCB).

3.1.2 Related work

Proxy caching for video has been explored in [84, 101, 43] under network conditions similar to those in Case 1 (QoS networks in Fig. 3.1a). Prefix caching, proposed by Sen *et al.* [84], is a special form of selective video caching, which involves caching

only a group of consecutive frames at the beginning of the video sequence to smooth and reduce the bit-rate of a VBR video. We will show that our proposed SCQ algorithm, compared to prefix caching, requires less client buffer B_c while achieving the same bandwidth reduction (Case 1), and that SCB can improve robustness more than prefix caching (Case 2). Note that when B_c is large, SCQ/SCB can reduce to prefix caching (see Sections 3.3 and 3.4.2).

Wang *et al.* propose a "video staging" algorithm in [101], which prefetches to the proxy a portion of bits from the video frames whose size is larger than a pre-determined "cut-off" rate, to reduce the bandwidth on the server-proxy channel. Therefore some frames are separated into two parts: one is cached on the proxy and the other remains on the server. By contrast, our proposed algorithms are *frame-wise* caching schemes so that a frame is either not available at the proxy or it is stored in its entirety. One advantage of frame-wise caching is that those frames available in the proxy can actually be played by the client (this would not be possible with partially cached frames) in the event where congestion prevents any data from being delivered from the server for some period of time. Another advantage of frame-wise caching scheme is that the proxy can easily add (or drop) more frames when cache space increases upon the changes of caching condition (such as network status, video object popularities, cache space, etc.), by using a caching table created off-line (proposed in Section 3.4.4). As an example, with a staging approach, if the popularity of a video increases the proxy will need to increase the

percentage of data in *all* frames (sending a request to the server to achieve this). Instead, with frame-wise caching only a few complete frames need to be requested from the server.

Ma *et al.* [43] also study a frame-wise video caching problem slightly different from that in Case 1, where selective caching is performed but the algorithm attempts to select groups of consecutive frames rather than isolated frames, in order to reduce the complexity of proxy management. In our work, by using a caching table, the proposed SCQ/SCB algorithms can select isolated frames (during iterations) to maximally improve the overall performance without increase the complexity for proxy online operations. Another major difference between our proposed work and other works in [43, 84, 101] is that we provide a caching strategy (SCB) for video delivery over best-effort networks, which are the most popular nowadays but are not explicitly considered in those works.

Both proposed SCQ/SCB algorithms consider non-layered coded video streams, while caching for layered (scalable) video can be found in [72, 38]. Rejaie *et al.* [72] propose a video caching algorithm for scalable video, which co-operates with the congestion control mechanism (for best-effort networks in Case 2) proposed in [69]. This work studies the caching replacement mechanism and cache resource allocation problem, according to the popularity of video objects, e.g., more layers of the video with higher popularity will be cached, and vice versa. Therefore the overall streaming performance can be improved (e.g., less network congestion, better

playback quality). Kangasharju *et al.* formulate the caching problem for layered video differently, aiming to maximize the overall revenue for the service providers [38]. Tewari *et al.* [91] and Reisslein *et al.* [67] study cache replacement of streaming media (in non-layered format) to improve the cache hit ratio and therefore the streaming quality. Our work can be complementary to [91, 67], as we are focusing on the problem of selecting which part of the video should be cached, after the cache space for this particular video has been allocated.

The rest of this chapter is organized as follows. Section 3.2 gives the background and definitions. Section 3.3 addresses the video caching problem for QoS networks and proposes the SCQ algorithm, while the SCB algorithm is proposed in Section 3.4 to solve the caching problem for best-effort networks. The experimental results are shown in Section 3.5. Finally, Section 3.6 concludes the chapter.

3.2 Basic definitions

Most standard video codecs (e.g., [52, 3]) produce VBR data after compression, which leads to high data burstiness. Usually there is a small start-up playback delay d (here we define it as the duration between sending out the first packet and playing the first frame) for most existing streaming video services to allow the client buffer to store a few beginning frames before playback starts. This delay is useful to (i) smooth the burstiness of VBR video data [78]; and (ii) to provide robustness against packet delay resulting from poor network conditions (in best-effort networks), thus

playback from the client buffer is possible even when frames are being delayed. For this reason, we always cache this beginning portion of video data, which is referred as the "required initial buffering segment" (I_{req}), such that the client can start to playback with a smaller start-up delay. When we can assign more cache space than I_{req} for a given video, we can choose between continuing to cache the immediately following frames (as would be done in a prefix caching technique), or instead selecting other intermediate frames to be cached. In this chapter, we consider the latter option, i.e., selective caching rather than prefix caching.

Assume there are N frames in a video V, denoted as $F(i), i = [1, 2, ..., N]$; each frame $F(i)$ has a size of $R(i)$ bytes, and a constant playback duration of T_F seconds (e.g., $T_F = 1/30$ *second*). We discretize the time axis t with intervals of T_F. The total size of the video is $R_{total} = \sum_{i=1}^{N} R(i)$. We define a "caching indicator sequence", $\mathcal{A} = [a(1), a(2), ..., a(N)]$, to indicate whether the i^{th} frame $F(i)$ is cached or not:

$$a(i) = \begin{cases} 0 & \text{if frame } F(i) \text{ is not cached} \\ 1 & \text{if frame } F(i) \text{ is cached} \end{cases} \tag{3.1}$$

A sequence $\mathcal{A}$ uniquely defines which part of video is cached, and can represent a selective caching scheme. We denote $\mathcal{A}_\phi$ (or simply ϕ) as the zero sequence, i.e., the

sequence where no frames are cached so that all $a(i) = 0$. A possible $\mathcal{A}$ must satisfy the cache space constraint

$$\sum_{i=1}^{N} R(i)a(i) \leq H, \quad a(i) \in \mathcal{A} \tag{3.2}$$

We also denote $\mathcal{A}^1$ as the indicator sequence where only I_{req} is cached. H is pre-determined for each video object.

3.3 Video caching in QoS networks

3.3.1 Problem formulation

An example of a QoS network is shown in Fig. 3.1a. We assume that a CBR bandwidth is reserved on the server-proxy backbone, and the cost of reservation is proportional to that bandwidth. Therefore we always reserve only the minimum required bandwidth for video delivery in a particular streaming session. Define C_r as the *required bandwidth* of a CBR channel to deliver the video V on time for real-time playback without jitter, given a finite start-up delay d. Due to the data burstiness of VBR video, C_r is usually higher than the *average video data rate*, R_{avg}, to avoid decoder buffer underflow. Feng *et al.* [14] proposed a general method to find such C_r for pre-coded video without caching (where C_r is defined as *critical bandwidth*) in [14]. We will show that C_r changes after some frames are cached, and is a function of caching indicator $\mathcal{A}$. One of our objectives in this case is to choose

$\mathcal{A}$ to minimize $C_r(\mathcal{A})$ by caching selected frames, so that the bandwidth reservation cost on server-proxy channel can be reduced.

However, while there may exist many possible $\mathcal{A}$ which lead to the same maximum reduction on $C_r(\mathcal{A})$, they may require different amount of client buffer size B_c to achieve that reduction, as shown in the analysis below. Considering both bandwidth and buffer size, we formulate the video caching problem for QoS networks as follows.

Problem formulation 1: *Given a limited proxy cache space H for a video sequence ($H < R_{total}$), a pre-encoded video stream V with N frames and a fixed delay d, among all possible $\mathcal{A}$ satisfying (3.2), find $\mathcal{A}^*$ which maximally reduces $C_r(\mathcal{A})$ after caching while requiring a minimum B_c to achieve that bandwidth reduction.*

3.3.2 Analysis on bandwidth reduction

To calculate $C_r(\mathcal{A})$, here we will extend the solution in [14] with minor modifications. We will only refer to the results from [14]; readers can refer to [14, 74, 83] for more details. Assume the transmission starts at time $t_0 = -d$, and playback starts at $t = 0$, the start-up delay is d ($d \geq 0$). Frame $F(i)$ is scheduled to be displayed at time $t = i$. Define $S_{\mathcal{A}}(t)$ as the *cumulative frame rate* at time t, for a given $\mathcal{A}$,

$$S_{\mathcal{A}}(t) = \sum_{i=0}^{t} (1 - a(i))R(i). \tag{3.3}$$

Define $C(t)$ as the server-proxy channel bandwidth at time t. A feasible channel rate function $C(t)$ to ensure a continuous playback must meet the following constraint:

$$\sum_{i=-d}^{t} C(i) \geq S_{\mathcal{A}}(t), \quad \text{for all} \quad t \in [1, N]. \tag{3.4}$$

$S_{\mathcal{A}}(t)$ and $\sum_{i=-d}^{t} C(i)$ can been thought of as the video data *consumption curve* at the client and the data *supply curve* from the channel, respectively. Since cached frames do not consume the server-proxy bandwidth and they can be fetched when needed (right before decoding) from the proxy, then the cached frames can be excluded from the client consumption curve in (3.3). Eq. (3.4) means that the supply curve should be greater than the consumption curve in order to avoid client buffer underflow (see Fig. 3.2). For a CBR channel, the bandwidth cannot exceed the constant allocated rate, so that

$$C(t) \leq C_r(\mathcal{A}), \quad \text{for all} \quad t \in [1, N]. \tag{3.5}$$

Define the *slope function* of a video sequence to be $L_{\mathcal{A}}(t)$,

$$L_{\mathcal{A}}(t) = S_{\mathcal{A}}(t)/t. \tag{3.6}$$

Fig. 3.2 shows an example of $S_{\mathcal{A}}(t)$ and $L_{\mathcal{A}}(t)$. In fact, $L_{\mathcal{A}}(t)$ represents the lower bound of a feasible $C(t)$, and C_r can be obtained as (see [14, 74, 83] for a proof)

$$C_r(\mathcal{A}) = \max_{1 \leq t \leq N} \{L_{\mathcal{A}}(t)\}, \quad t \in [1, N], \tag{3.7}$$

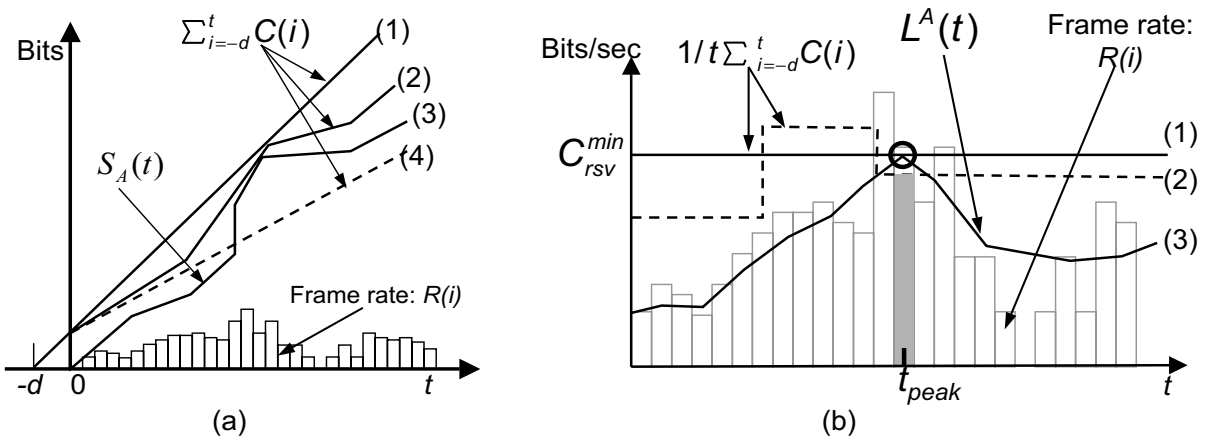

Figure 3.2: Cumulative rate and slope functions. (a): Cumulative frame/channel rate. Curve (1), (2) and (4) are cumulative channel rate functions, $\sum_i C(i)$. Among them only (1) and (4) represent CBR channels. Curve (3) is the cumulative frame rate, $S_{\mathcal{A}}(t)$. Curve (1) and (2) are feasible channel rate, while (4) is not. (b): Slope function $L_{\mathcal{A}}(t)$ is drawn in curve (3). The minimum CBR channel bandwidth that has to be reserved is $C_r = \max\{L_{\mathcal{A}}(t)\}$.

i.e., the minimum bandwidth $C_r(\mathcal{A})$ that has to be reserved on a CBR channel is the maximum value of the video slope function $L_{\mathcal{A}}(t)$, which is reached at time t_{peak}, referred to as the *consumption peak time*,

$$t_{peak} = \arg \max_{1 \le t \le N} \{L_{\mathcal{A}}(t)\}. \tag{3.8}$$

Suppose we want to cache one more frame $F(k)$ after some frames have already been cached. Let $\mathcal{A}$ and $\mathcal{A}_k$ be the caching indicator sequences before and after caching $F(k)$, respectively. The following proposition shows the change in bandwidth after caching $F(k)$.

Proposition 1 For a given $F(k)$ and $\mathcal{A}$, $\max\{L_{\mathcal{A}_k}(t)\} = \max\{L_{\mathcal{A}}(t)\} - \Delta l$, where

$$\Delta l = \begin{cases} R(k)/t_{peak}, & k \in [1, t_{peak}], \\ 0, & \text{otherwise.} \end{cases} \tag{3.9}$$

See Appendix for a proof. An illustration is shown in Fig. 3.3. This means that $\max\{L_{\mathcal{A}_k}(t)\}$ can be reduced if and only if $k \in [1, t_{peak}]$. Therefore we should cache a frame *before* t_{peak} to reduce C_r, and the reduction Δl depends only on the size of the cached frame but not on its position, i.e.,

$$C_r(\mathcal{A}_k) = C_r(\mathcal{A}) - R(k)/t_{peak}, \text{ if } k \in [1, t_{peak}]. \tag{3.10}$$

The above conclusion indicates that caching successive frames from the beginning of the video sequence is one of the caching schemes that can reduce C_r maximally. As an example, "prefix caching", which selects the group of beginning frames to be cached until the cache space is full, is a good approach to reduce the bandwidth requirements [84].

3.3.3 Client buffer analysis

Recall that we need to select frames to be cached to (i) *maximally reduce* $C_r(\mathcal{A})$, and (ii) *require the minimum* B_c *to achieve that reduction.* Selecting different frames to be cached within the period of $[1, t_{peak}]$ leads to different requirements of B_c. This can be explained by the following buffer analysis. Define a byte-level buffer trace

function, $B_{\mathcal{A}}(t)$, as the amount of video data (in number of bytes) in the client buffer during playback (given a cache sequence $\mathcal{A}$), which can be written as (3.11) (see [14]),

$$B_{\mathcal{A}}(t) = \sum_{i=-d}^{t} C(i) - S_{\mathcal{A}}(t), \tag{3.11}$$

where $C(i) \leq C_r(\mathcal{A})$ in (3.7). The required client buffer size, $B_{max}(\mathcal{A})$, is the maximum value of $B_{\mathcal{A}}(t)$. Thus we need to minimize $B_{max}(\mathcal{A})$ in order to reduce the required client buffer size to achieve the reduction of $C_r(\mathcal{A})$ stated in Proposition 1. We also denote t_{max} as the *buffer peak time*, when $B_{\mathcal{A}}(t)$ reaches to $B_{max}(\mathcal{A})$,

$$t_{max} = \arg \max_{1 \leq t \leq N} \{B_{\mathcal{A}}(t)\}. \tag{3.12}$$

We first examine the change from $B_{\mathcal{A}}(t)$ to $B_{\mathcal{A}_k}(t)$ for the period of $1 \leq t \leq t_{peak}$, after caching one frame $F(k)$ (note that $1 \leq k \leq t_{peak}$ according to Proposition 1).

Because the channel bandwidth is constant, see (3.4) and (3.5), we have $C(t) = C_r(\mathcal{A})$ for $1 \leq t \leq t_{peak}$ (so that $C(t) = C_r(\mathcal{A}_k)$ after caching $F(k)$). From (3.11) and (3.10) we have

$$\begin{aligned} B_{\mathcal{A}_k}(t) &= C_r(\mathcal{A}_k)t - S_{\mathcal{A}_k}(t) \\ &= C_r(\mathcal{A})t - R(k)\frac{t}{t_{peak}} - S_{\mathcal{A}_k}(t), \end{aligned} \tag{3.13}$$

where $1 \leq t \leq t_{peak}$. Note that after caching $F(k)$, $S_{\mathcal{A}_k}(t) = S_{\mathcal{A}}(t)$ for $1 \leq t < k$; and $S_{\mathcal{A}_k}(t) = S_{\mathcal{A}}(t) - R(k)$ for $k \leq t \leq t_{peak}$. Thus we have

$$B_{\mathcal{A}_k}(t) = \begin{cases} B_{\mathcal{A}}(t) - R(k)\frac{t}{t_{peak}}, & 1 \leq t < k, \\ B_{\mathcal{A}}(t) - R(k)\frac{t}{t_{peak}} + R(k) & k \leq t \leq t_{peak}. \end{cases} \tag{3.14}$$

Eq. (3.14) means that after caching frame $F(k)$, the new buffer trace *decreases* before k (due to the reduction of C_r), and *increases* during $[k, t_{peak}]$ (due to the removal of $R(k)$ from the consumption curve)[2]. It also indicates that at any particular time $t \in [1, t_{peak}]$, the amount of increment/decrement of $B_{\mathcal{A}_k}(t)$ depends only on $R(k)$. More specifically, if $t_{max} < t_{peak}$, we can reduce $B_{max}(\mathcal{A}_k)$ *only* if we cache frame $F(k)$ after t_{max}, and we will have

$$B_{max}(\mathcal{A}_k) = B_{max}(\mathcal{A}) - \Delta b, \qquad \text{if } t_{max} < k \leq t_{peak}, \tag{3.15}$$

$$\text{where} \qquad \Delta b = \frac{t_{max}}{t_{peak}} R(k). \tag{3.16}$$

The change of $B_{\mathcal{A}_k}(t)$ for the remaining period $t_{peak} \leq t \leq N$ may not be easily expressed in a closed form. However, based on the results from [14, 74, 83] we can find that the *maximum* buffer occupancy may increase (or at least not decrease) for the period of $t_{peak} < t \leq N$. This is because $B_{\mathcal{A}_k}(t_{peak}) = B_{\mathcal{A}}(t_{peak}) = 0$, and

[2]Eq. (3.14) is obtained as the delay of transmitting $F(k)$ (over the proxy-client channel) is small, and the delay is ignored to simplify the analysis. The frames after $F(k)$ are received earlier than they would have if caching is not used.

the only difference is that a smaller bandwidth ($C_{\mathcal{A}_k}(i)$) is available to transmit the video data for $t > t_{peak}$, after caching frame $F(k)$ (see [14] for details). Proposition 1 and (3.14) assume that t_{peak} or t_{max} do not change after $F(k)$ is cached. However those results still hold when t_{peak} or t_{max} changes, except that the absolute values of Δl and Δb will become smaller, which would not affect the conclusions in next section.

3.3.4 Proposed SCQ algorithm

The results in (3.10) and (3.15) lead to the following conclusion for caching one frame $F(k)$: *one can cache the frame at t_{peak}, $F(t_{peak})$, in order to minimize $C_r(\mathcal{A})$ and $B_{max}(\mathcal{A})$*. The reasons are:

1. From Proposition 1 we have to cache a frame before t_{peak} to reduce C_r.

2. From (3.14) and (3.15), if B_{max} is reached before t_{peak} (i.e., $t_{max} < t_{peak}$), caching the frame $F(t_{peak})$ can also reduce B_{max}. More specifically, in this case, caching any frame within the period of $(t_{max}, t_{peak}]$ has the *same* effect in terms of reducing both B_{max} and C_r. Similarly, caching any frame of a given size within the period of $[1, t_{max}]$ has the *same* effect on reducing C_r while *increasing* B_{max}. See Fig. 3.4 for an illustration. Therefore, simply selecting frame $F(t_{peak})$ is guaranteed to reduce B_{max} and C_r, without requiring to compute t_{max}.

3. If B_{max} is reached *after* t_{peak} (i.e., $t_{max} > t_{peak}$), caching any frames before t_{peak} can not reduce (or may increase) B_{max}. However, from (3.14), we can see that caching frame $F(t_{peak})$ has the effect of reducing $B_{\mathcal{A}}(t)$ for the longest duration of $[1, t_{peak}]$. Also as more frames are cached, t_{peak} tends to increase monotonically. Once $t_{max} < t_{peak}$, we will have the above situation 2, and the new B_{max} is already reduced if we keep on caching frame $F(t_{peak})$.

The above analysis shows that the changes on $C_r(\mathcal{A})$ and $B_{max}(\mathcal{A})$ depend *only* on the cached frame size (i.e., the absolute values of Δl and Δb *depend only* on $R(k)$), not the exact position of the cached frame, as long as that frame falls in the range of $[1, t_{peak}]$.

For a VBR video, the frames around t_{peak} may have different sizes, the search for the optimal selection of frames to be cached can be complex when cache space H is large. Therefore we propose an heuristic approach, SCQ, for selective video caching in QoS networks, which selects frames iteratively. During each iteration, SCQ computes the $L_{\mathcal{A}}(t)$ and locates t_{peak} using (3.8), then selects the frame at t_{peak} (i.e., $F(t_{peak})$) to be cached. This process is iterated until the cache space is full. The detailed procedure is summarized in the following steps.

Step 1. Initialization. Set $n = 1$ (n is the iteration index). Cache the required initial segment I_{req} and set $\mathcal{A}^1$ correspondingly (see Section 3.2). Let $\mathcal{A}^n$ be the cache indicator after the n^{th} iteration. Set $H \Leftarrow H - I_{req}$.

Step 2: Find $t^n_{peak} = \arg\max_t\{L_{\mathcal{A}^n}(t)\}$ for the n^{th} iteration.

Step 3: Cache frame $F(t^n_{peak})$, set $a(t^n_{peak}) = 1$; set $H \Leftarrow H - R(t^n_{peak})$.

Step 4: If there is no cache space left ($H \leq 0$), procedure ends. Otherwise, set $n \Leftarrow n + 1$, go to Step 2 (start next iteration).

We will obtain the solution $\mathcal{A}^* = \mathcal{A}^n$ at the last iteration. Note that usually $t_{max} > t_{peak}$ during the initial iterations. Therefore increasing of B_{max} can not be avoided since we have to cache a frame before t_{peak} to reduce bandwidth. B_{max} starts to decrease once $t_{max} < t_{peak}$ after more frames are cached, and SCQ tries to keep that initial increment as small as possible. See the results in Section 3.5.

3.4 Video caching in best-effort networks

An example of video caching in best-effort networks is shown in Fig. 3.1b. The server-proxy channel bandwidth may have variations due to network congestion or other poor conditions. These variations can cause dramatic degradation on continuous video playback quality, since packets that arrive too late are considered to be lost. Thus it is useful for the client to buffer a certain number of frames before and during playback, in order to increase the likelihood that frames are available for playback in the decoder buffer during the periods of packet lost (delay). The more frames are buffered at a given time, the more *robustness* there will be against the packet delay.

However, the frames in VBR video have different sizes, which means that the number of buffered frames in the client's buffer may not be *constant* during playback.

Periods during which the number of frames in the decoder buffer becomes low are referred to as the *risky periods* (less robust). Our caching goal here is to improve the robustness by increasing the number of frames in the client buffer during risky periods.

As we are focusing on an off-line caching algorithm that only has the knowledge of the video sequence, we do not make any assumptions about the actual bandwidth variations while deriving the caching algorithm. The risky periods of the video sequence can be located before transmission by analyzing the decoder buffer contents during a "virtual" playback, where a constant server-proxy bandwidth C (close to the average video bit-rate) is applied. Note that the network congestion may happen any time during a real-time session. However the congestion is more likely to cause quality degradation when the client buffer contains fewer frames, i.e., during the risky periods identified in our analysis. In the simulation, we transmitted video (with partial caching) using a server-proxy channel with bandwidth variations to verify the effectiveness of the buffer analysis and the caching algorithm.

We define a frame-level *buffer trace* function, $B^f(t)$, which indicates the number of *frames available at the client* during the playback. A measurement of the robustness of a video stream U can be defined as

$$U = \min_t \{B^f(t)\}, \tag{3.17}$$

i.e., the minimum value (referred to as a *trough*) of the buffer trace in number of frames. Risky period is the time when a trough occurs, i.e., $t_r = \arg\min_t\{B^f(t)\}$. There might be many risky periods/frames in one session. The larger $B^f(t)$, the more robust this video stream will be around time t.

The robustness metric U defined in (3.17) corresponds to using a *MaxMin* criterion (as we will try to maximize U by caching frames, see below). Obviously there are alternative ways to define robustness such as, for instance, the average number of frames in the buffer (referred to as a *MaxAverage* criterion):

$$U_a = \frac{1}{N}\sum_{t=1}^{N} B^f(t). \tag{3.18}$$

Each of these two measures of robustness, U or U_a, leads us to different algorithms to select which frames should be cached. For most scenarios in this chapter, we use the MaxMin criterion for robustness, and we will use the MaxAverage criterion only to break the tie among multiple choices that all improve U in the same way (see Section 3.4.3).

Note that $B^f(t)$ (measured in number of frames) is used to calculate robustness; while $B(t)$ (measured in number of bits, see (3.11)) is used to determine the occupancy of the client physical buffer during playback. It should be emphasized that *both* the frames in the client's physical buffer and the cached frames at the proxy are counted as the *available* frames for the client. In other words, cached frames can

increase $B^f(t)$ (and therefore the robustness), while not occupying client physical buffer, i.e., they do not increase $B(t)$[3].

3.4.1 Problem formulation

In this case the decoder buffer size B_c is the bottleneck in improving U. For example, a straightforward way to improve U is to cache the "earlier" frames from the beginning of video sequence. Thus the client can retrieve the "later" non-cached frames from the server, while it is displaying the cached frames retrieved from proxy. At the time when those "later" frames that are not cached start to be displayed, many of them are already buffered at the client, and therefore U is improved. However, this method could soon fill up the client buffer since the frames retrieved from the server are not played until all the cached frames (scheduled to be displayed at earlier time) are displayed. Thus the server may have to slow down the transmission speed when the client buffer is full, which waste the bandwidth to further increase the robustness. Therefore a proper selection scheme should be designed under the constraint a limited memory buffer B_c. We formulate the caching problem as follows (assume B_{max} is known).

Problem formulation 2: *Given a limited cache space ($H < R_{total}$) on the proxy, a pre-encoded video stream V with N frames and a fixed delay d, among all possible $\mathcal{A}$ satisfying (3.2), find the cache indicator sequence $\mathcal{A}^*$ such that the*

[3]Again, this is true because the cached frames can be fetched quickly from proxy right before their playback time, such that they can be excluded from the consumption of client physical buffer, see explanation in Section-3.1.

robustness $U = \min_t\{B^f_{\mathcal{A}^*}(t)\}$ *is maximized, without exceeding the maximum client buffer size* B_{max}.

3.4.2 Analysis on buffer trace after caching

The byte-level buffer trace function $B(t)$ can be computed from (3.11). The frame-level buffer trace function $B^f(t)$ can be obtained by simulating the transmission frame by frame, assuming that the nominal channel rate C is provided. An example is shown in Fig. 3.5.

We first study the case of caching a single frame. For a given $\mathcal{A}$, and available channel bandwidth C, if the client buffer size (B_{max}) is large enough, i.e., $B_{max} \geq \max_t\{B_{\mathcal{A}}(t)\}$, then based on (3.11) we have

$$\begin{aligned} B_{\mathcal{A}}(t) &= C(t+d) - S_{\mathcal{A}}(t) \\ &= C(t+d) - \left(\sum_{i=1}^{t} R_i - \sum_{i=1}^{t} a(i)R(i)\right) \\ &= B_{\phi}(t) + \sum_{i=1}^{t} a(i)R(i), \end{aligned} \tag{3.19}$$

where $B_{\phi}(t)$ is the buffer trace (in bits) where no frame is cached ($\mathcal{A} = \phi$). From (3.19) it can be seen that *caching one frame* $F(k)$ *increases the buffer trace* $B_{\mathcal{A}}(t)$ *for the duration* $t \in [k, N]$ *by the cached frame size* $R(k)$. This is because, after being cached, frame $F(k)$ will be fetched from the proxy rather than from the server, thus all the non-cached frames later than $F(k)$ are "shifted" to an earlier transmission

time (because frame $F(k)$ is skipped for transmission) from server. Those later frames (after $F(k)$) will stay in the client buffer for a longer period and thus increase $B_{\mathcal{A}}(t)$ (and corresponding $B^f_{\mathcal{A}}(t)$), by caching $F(k)$.

A simple example of the "raise" in $B_{\mathcal{A}}(t)$ after caching $F(k)$ is shown in Fig. 3.6. However, (3.19) does not hold if there is a tight buffer size limitation, i.e., $B_{max} < \max_t\{B_{\mathcal{A}}(t)\}$. If the decoder buffer is full, then the server and proxy have to reduce the transmission speed, which means $C(i)$ is smaller than C, otherwise packets will be discarded due to client buffer overflow.

Proposition 2 If there exists a t_{max} (defined in (3.12)) such that $B_\phi(t_{max}) = B_{max}$ (where B_{max} is the known maximum buffer size), then caching one frame $F(t_i)$ which is located before t_{max} ($t_i < t_{max}$) can only increase the buffer trace by approximately $R(t_i)$ between time $\hat{t_i}$ and t_{max}, where $\hat{t_i}$ is the transmission time of frame $F(t_i)$, and $\hat{t_i} < t_i < t_{max}$.

$$B_{\mathcal{A}}(t) = \begin{cases} B_\phi(t) & \text{if } t \le \hat{t_i}\ , \\ B_\phi(t) + R(t_i) & \text{if } \hat{t_i} < t \le t'_{max}\ , \\ B_{max} & \text{if } t'_{max} < t \le t_{max}\ , \\ B_\phi(t) & \text{if } t_{max} < t \le N. \end{cases} \tag{3.20}$$

See the Appendix for a proof. In short, before $B_{\mathcal{A}}(t)$ reaches B_{max} (i.e., $t_i < t < t'_{max}$), it is "lifted" according to (3.19), and as a result $B_{\mathcal{A}}(t)$ will reach B_{max} at an "earlier" time t'_{max} (where $t'_{max} < t_{max}$). Then the proxy and/or server have to reduce transmission speed after time t'_{max} when the buffer is full, until it is drained

to accept further data. This reduction cancels out the "extra" frames cumulated in the buffer, so that the remaining buffer trace for $t > t_{max}$ is the same as if no frames are cached. See Fig. 3.7 for an illustration.

Caching multiple frames is similar to the single frame case. When additional frames are cached, the buffer trace $B(t)$ is "raised" consequently, and may hit the maximum bound B_{max} at more points. Therefore, we conclude that *each cached frame can only increase the buffer trace between its scheduled transmission time* $\hat{t}_i$ *and next nearest* t_{max}*, if it exists, where* $B(t_{max}) = B_{max}$.

3.4.3 Proposed SCB algorithm

Based on the above conclusions, we now propose the Selective Caching for Best-effort networks (SCB), which iteratively selects one frame that is located within the range of $[t_{max}, t_r]$, where t_{max} is the closest buffer peak time before t_r. An example of SCB is shown in Fig. 3.8, where troughs occurs at t_1 and t_4 before caching. According to Proposition 2, caching frames before the buffer peak time t_2 will not lift $B^f(t)$ after t_2. So after caching frames before before t_1 to increase $B^f(t_1)$, we should select frames between t_2 and t_4 to increase $B^f(t_4)$ (see Fig. 3.8(b)), and therefore improve U defined in (3.17). The details of SCB algorithm are summarized in the following steps.

Step 1. Initialization. Same as Step 1 in SCQ algorithm. See Section 3.3.4.

Step 2. In the n^{th} iteration, find the most *risky* period $t_r^n = \arg\min_t\{B_n^f(t)\}$, e.g., time t_1 in Fig. 3.8a. If there are multiple t_r^n, choose the first one (the MaxMin criteria is applied first, and MaxAverage is applied to break the tie choices if needed).

Step 3. Find the *nearest* buffer peak time t_{max}^n before the chosen t_r^n (e.g., t_2 for risky time t_4 in Fig. 3.8b). If no maximum peak exists before t_r^n, set $t_{max}^n = 0$ (e.g., $t = 0$ for t_1 in Fig. 3.8a). Note that t_{max}^n is obtained from the byte-level buffer trace $B_{\mathcal{A}^n}(t)$.

Step 4. Select *one* frame $F(c^n)$ which is right after t_{max}^n (obtained in Step 3) to be cached, set the $a(c^n) = 1$. Update the trace $B_{\mathcal{A}^n}^f(t)$ and set $H \Leftarrow H - R(c^n)$. If $H \leq 0$, there is not enough space left on proxy, procedure ends. Otherwise, set $n \Leftarrow n + 1$, go to Step 2.

In each iteration, we first locate the t_r for U, thus to increase U is the same as to increase $B^f(t_r)$. From Proposition 2, we know that in order to increase $B^f(t_r)$, we have to cache the frames *after* the nearest previous buffer peak time.[4] Since there might be multiple choices for selecting, e.g., caching any frames between t_{max} and t_r can increase $B^f(t_r)$, the MaxAverage criteria for robustness requires to select the

[4] Our goal is to increase the number of available frames in the buffer during the playback, which is measured by $B^f(t)$. The exact value of $B^f(t)$ may have a slightly different shape from $B(t)$, due the variable size of frames in VBR video. However, the increment in $B(t)$ should also lead to the increment in $B^f(t)$ when there is more data in the client buffer. Therefore the results in (3.19) and (3.20) can also be applied to approximate the increment in $B^f(t)$ after caching a frame $F(k)$.

frame furthest away from t_r but after the nearest previous peak, t_{max}. This provides the largest increase in average robustness U_a.

3.4.4 Caching table

Both SCQ/SCB algorithms select *additional* frames when there is more cache space available, while the frames selected earlier still remain being cached. Therefore we can perform the SCQ/SCB algorithms for the complete video sequence to determine the *order* of all frames to selected for caching (this can be achieved by setting the cache space equal to the size of the video), and store the results into a *cache table*. When the available cache space increases or decreases, then the proxy can add or remove frames that need to be cached according to the cache table, without re-computing the selection procedure. Thus, the caching scalability and low complexity (for online operations of the proxy) can be achieved, by using such a cache table.

3.5 Experimental results

Experimental results on the SCQ algorithm described in Section 3.3.4 are shown in Fig. 3.9. A video clip (part of movie *Star Wars* in MPEG-1 [29]) with 10,000 consecutive frames is used for simulation. The original video has the average rate (R_{avg}) of 5,516 KBits/sec, (peak frame rate is 9,812 KBits/sec). The total size of video clip (R_{total}) is 280.6 MBytes. Fig. 3.9a shows that the proposed SCQ algorithm can reduce the server-proxy bandwidth almost the same as that of the prefix caching

algorithm. Fig. 3.9b shows that as expected, the SCQ algorithm requires a much smaller client buffer size to achieve the same bandwidth reduction as prefix caching.

In the experiments of SCB algorithm, the video clip having 10,000 frames is encoded with MPEG-2 under rate control[5] similar to [55]. The original video data has an average rate (R_{avg}) of 2,112 KBits/sec, with peak data rate 2,400 KBytes/sec. The average frame size is 88 KBits. The average channel bandwidth, C, is also 2,112 KBits/sec. The client buffer size (B_c) is 512 KBytes. Fig. 3.10a shows the robustness $U = \min_t\{B^f(t)\}$ with respect to the percentage of video being cached. Fig. 3.10b shows the average number of frames in the buffer during the playback, or $U_a = \frac{1}{N}\sum_{t=1}^{N} B^f(t)$. Note that as defined in Section 3.4.1, the selective caching uses the MaxMin robustness criterion first rather than the MaxAverage criterion to eliminate the worst case first. When only a portion of the video is cached, the SCB outperforms prefix caching in both cases (for U and U_a).

Fig. 3.11 shows the simulation results to verify the effectiveness of our definition of robustness (U and U_a). We simulate the delivery of streaming video when it is partially cached. The cached part of the video is sent to client from the proxy with no loss. The non-cached frames are retrieved from server (via proxy), starting from the beginning of the playback session, and the data is buffered at client until it is displayed. The client physical buffer size is 512KB.

[5]The reason to apply rate control is to avoid large variance of video data rate to avoid potential network congestion in a best-effort network.

We use a binary erasure channel (BEC) [20] to model the server-proxy channel. A packet (we use a fixed packet size of 512 bytes) is lost with probability ϵ, and arrives to the client correctly with probability $1-\epsilon$, where ϵ is the channel packet loss rate. All lost packets are recovered by retransmission. In this simulation all frames *have to be played*, thus if a frame arrives too late to the client, due to delay or retransmission, the previous frame is "frozen" on the screen until it arrives. We refer to the period during which a frame is "frozen" as the "jitter duration" T_J. The fraction of T_J/T_V (where T_V is the total video playback time) is used to measure the continuity of the playback. $T_J/T_V = 0$ means that the playback has no jitter; a larger T_J/T_V indicates that more jitter happens during the playback. Thus a smaller T_J/T_V indicates more robustness for a continuous playback. The experiment uses 1000 realizations, the packet loss is performed randomly with the given channel error ϵ. We can see from Fig. 3.11a shows that when a larger proportion of the video is cached, the robustness is increased and the jitter duration is reduced. Fig. 3.11b shows the jitter duration with different packet loss rate. In both cases, the proposed SCB algorithm leads to smaller jitter duration than prefix caching since it select frames to be cached to maximize the robustness.

In the presence of network congestion, the congestion duration can last over some unpredictable time-scale. During the congestion all packets are delayed, and the bandwidth drops to zero. Fig. 3.11c shows the results when both the channel congestion and the congestion duration (denoted by d_c) occurs randomly. d_c follows

an exponential distribution with mean of m_c. We test the robustness with two cache schemes with different value of m_c. Fig. 3.11c shows that in both schemes, the jitter duration increases when m_c becomes large. This is true as jitter is more likely to happen when network congestion becomes severe (last longer). As expected, the proposed SCB algorithm outperforms prefix caching algorithms with different m_c. The results showed in Fig. 3.11 verify the effectiveness of our definition of robustness criteria developed for SCB algorithm.

3.6 Conclusions

In this chapter, two novel approaches for proxy caching of video are presented for both the QoS networks and best-effort networks (e.g., the Internet). The video caching performance is measured differently in these network environments, i.e., for QoS networks, the metric of interest is the network bandwidth cost; while the robustness of continuous playback against poor network conditions is more important in best-effort networks. Therefore the caching algorithms should be designed accordingly. We also emphasized that some resources, such as client decoder buffer size and limited proxy cache space, are also critical for the design of video caching algorithms. We proposed two caching algorithms, SCQ and SCB, for QoS and best-effort networks, respectively. SCQ can reduce the network cost of bandwidth reservation near optimally and requires a small client buffer size to achieve it; SCB can increase the playback robustness while not violating the client buffer size budget. Both SCQ

and SCB algorithms provide good scalability for proxy space adjustment, and low complexity for proxy online operations. The proxy can easily reduce/increase the cache space for a video object while still maintain the good performance provided by these algorithms.

Both SCQ and SCB algorithms are designed for caching a single video object with a pre-allocated cache space. In the situation that the total cache space is limited, to determine how much cache space allocated to each video to maximize the overall performance can be an interesting resource allocation problem (e.g., [88, 67, 38, 75, 91].). The proposed caching algorithms (SCQ and SCB) in this Chapter are independent of any other cache space allocation mechanisms, and can be used in conjunction with them. The cache table described in Section 3.4.4 can be used to find the trade-offs between the cache space and the caching performance for each individual video sequence.

Appendix

Proof of Proposition 1.

From (3.3) and (3.6) we get

$$L_{\mathcal{A}_k}(t) = \begin{cases} L_{\mathcal{A}}(t), & 1 \le t < k \\ L_{\mathcal{A}}(t) - R(k)/t, & k \le t \le N \end{cases} \tag{3.21}$$

Recall the definition of t_{peak} in (3.8). If $k \le t_{peak}$, from the second case in (3.21), we have $\max\{L_{\mathcal{A}_k}(t)\} < \max\{L_{\mathcal{A}}(t)\}$, therefore caching a frame $F(k)$ before t_{peak} can reduce the required bandwidth C_r. The reduction is $\Delta l = R(k)/t_{peak}$ (note that this assumes t_{peak} remains the same after caching $F(k)$; however the Proposition still holds if t_{peak} changes except that the absolute value of Δl is smaller than $R(k)/t_{peak}$). Obviously if $k > t_{peak}$, caching that frame cannot reduce the maximum of $L_{\mathcal{A}}(t)$ which occurs before k.

Proof of Proposition 2.

From (3.19), we know that without the physical buffer size constraint, $B_{\mathcal{A}}(t)$ would exceed B_{max}. Thus to prevent the buffer overflow, the server and/or proxy has to reduce the transmission speed ($C(i)$) at (or before) t_{max}. However we assume the reduction of $C(i)$ is as small as possible[6] so that client buffer is always kept full

[6]This can be easily achieved by sending feedback from client to proxy indicating the buffer fullness during the transmission, so that the proxy/server can adjust the $C(i)$ accordingly.

during the period around t_{max}[7]. Therefore, at time t_{max}, $B_{\mathcal{A}}(t)$ also reaches the maximum, $B_{\mathcal{A}}(t_{max}) = B_{max}$.

Because $R(t_i) > 0$ and $t_{max} \geq \hat{t_i}$, it is easy to see that there must exist $t'_{max} < t_{max}$ such that $B_{\mathcal{A}}(t'_{max}) = B_{max}$. This means $B_{\mathcal{A}}(t)$ reaches B_{max} at an earlier time t'_{max} due to the increase of $R(t_i)$.

For $\hat{t_i} \leq t < t'_{max}$, we know from (3.19),

$$B_{\mathcal{A}}(t) = B_{\phi}(t) + R(\hat{t_i}), \tag{3.22}$$

For $t'_{max} < t \leq t_{max}$, $B_{\mathcal{A}}(t)$ remains full at B_{max} (as the above assumption). We also have $B_{\mathcal{A}}(t_{max}) = B_{\phi}(t_{max}) = B_{max}$ For $t > t_{max}$, from (3.11) we know that

$$\begin{aligned}
B_{\mathcal{A}}(t_{max}+1) &= \sum_{i=1}^{t_{max}+1} C(i) - \sum_{i=1}^{t_{max}+1} R(i) \\
&= \Big(\sum_{i=1}^{t_{max}} C(i) - \sum_{i=1}^{t_{max}} R(i) \Big) + C(t_{max}+1) - R(t_{max}+1) \\
&= B_{\mathcal{A}}(t_{max}) + C(t_{max}+1) - R(t_{max}+1) \\
&= B_{max} + C(t_{max}+1) - R(t_{max}+1) \\
&= B_{\phi}(t_{max}) + C(t_{max}+1) - R(t_{max}+1) \\
&= B_{\phi}(t_{max}+1)
\end{aligned}$$

[7]To always keep the buffer as full as possible is to maximize the robustness, by storing more frames in the buffer.

Applying this recursively for all $t_{max} < t \leq N$, we can get

$$B_{\mathcal{A}}(t) = B_{\phi}(t), \quad \text{where} \quad t_{max} < t \leq N. \tag{3.23}$$

Finally, for $t \leq \hat{t_i}$ ($\hat{t_i}$ is the transmission time of frame $F(t_i)$), obviously there is no change for the buffer trace, $B_{\mathcal{A}}(t) = B_{\phi}(t)$. Combining these results we get (3.20).

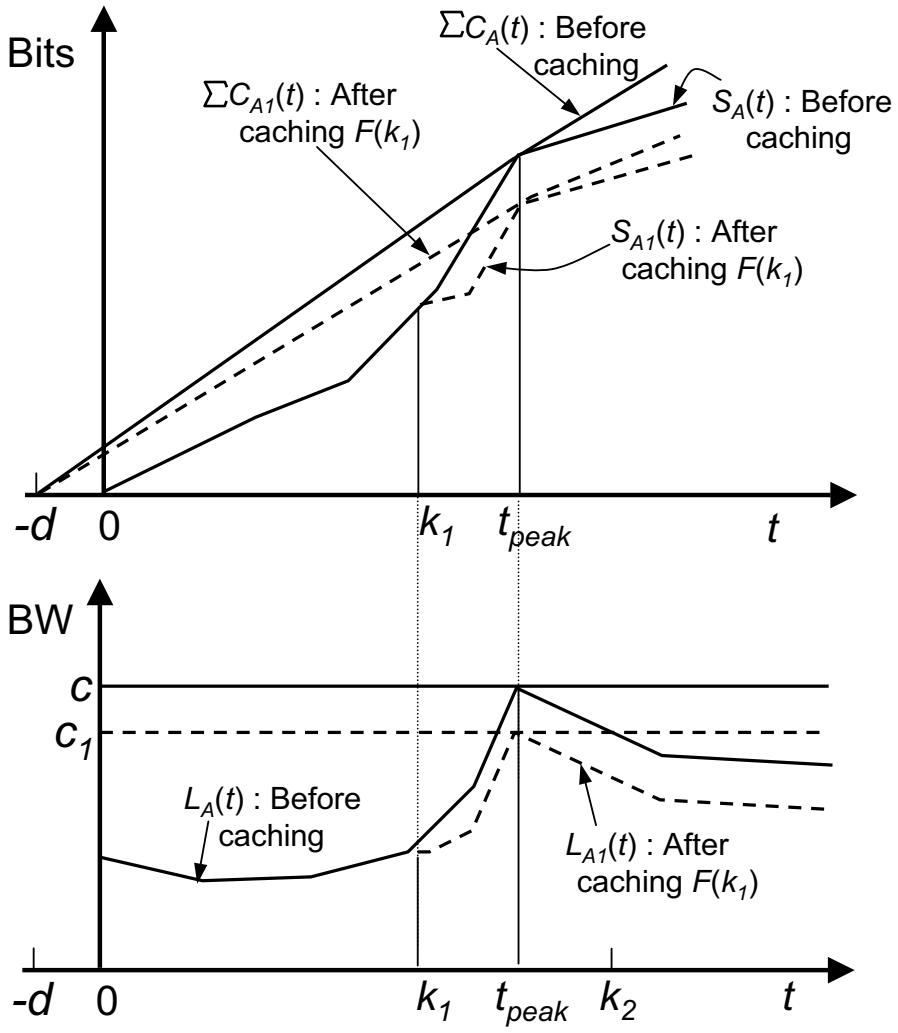

Figure 3.3: Illustration of Proposition 1. The lower figure shows the slope functions before and after caching one frame $F(k_1)$, represented by $L_{\mathcal{A}}(t)$ (solid line) and $L_{\mathcal{A}_1}(t)$ (dashed line), respectively. The upper figure is the corresponding cumulative channel/frame rate functions. After caching an "earlier" frame before t_{peak}, i.e., $k_1 \leq t_{peak}$, $\max\{L_{\mathcal{A}}(t)\}$ reduces from c to c_1. Obviously, caching a "later" frame $F(k_2)$, i.e., $k_2 > t_{peak}$, can not reduce $\max\{L_{\mathcal{A}}(t)\}$, which occurs at t_{peak}.

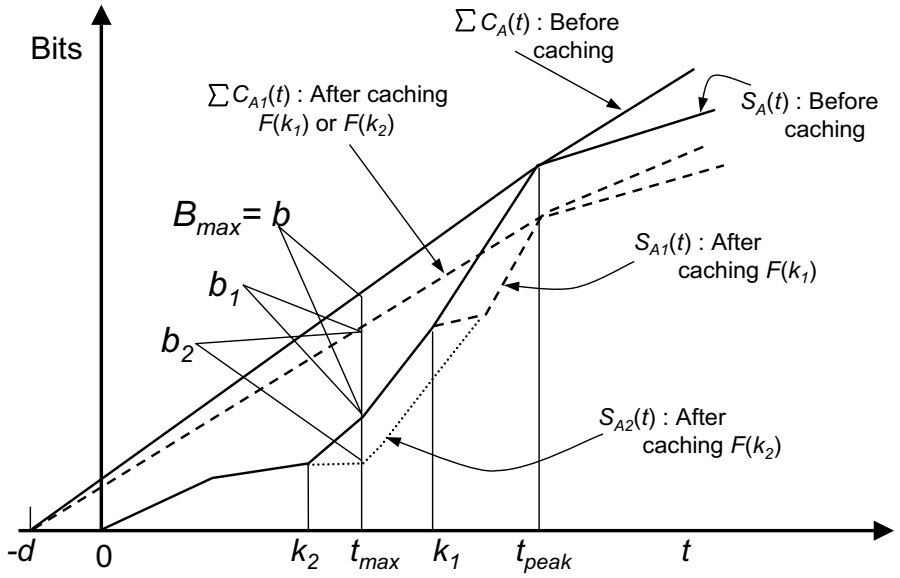

Figure 3.4: Illustration of caching one frame before or after t_{max}. $S_{\mathcal{A}_1}(t)$ and $S_{\mathcal{A}_2}(t)$ are the cumulative frame rate functions after caching $F(k_1)$ and $F(k_2)$, (drawn in dashed and dotted lines) respectively. Note that only *one* frame is selected in each case. If $R(k_1) = R(k_2)$ and $k_2 < t_{max} < k_1 < t_{peak}$, Proposition 1 shows that selecting $F(k_1)$ or $F(k_2)$ leads to the same reduction on bandwidth C_r. However, the corresponding changes in B_{max} are different: caching $F(k_1)$ reduces B_{max} from b to b_1 ($\Delta b = b_1 - b < 0$); while caching $F(k_2)$ increases B_{max} from b to b_2 ($\Delta b = b_2 - b > 0$). Therefore caching a frame between t_{max} and t_{peak} (e.g., $F(k_2)$) can reduce B_{max} while keeping the same reduction on C_r.

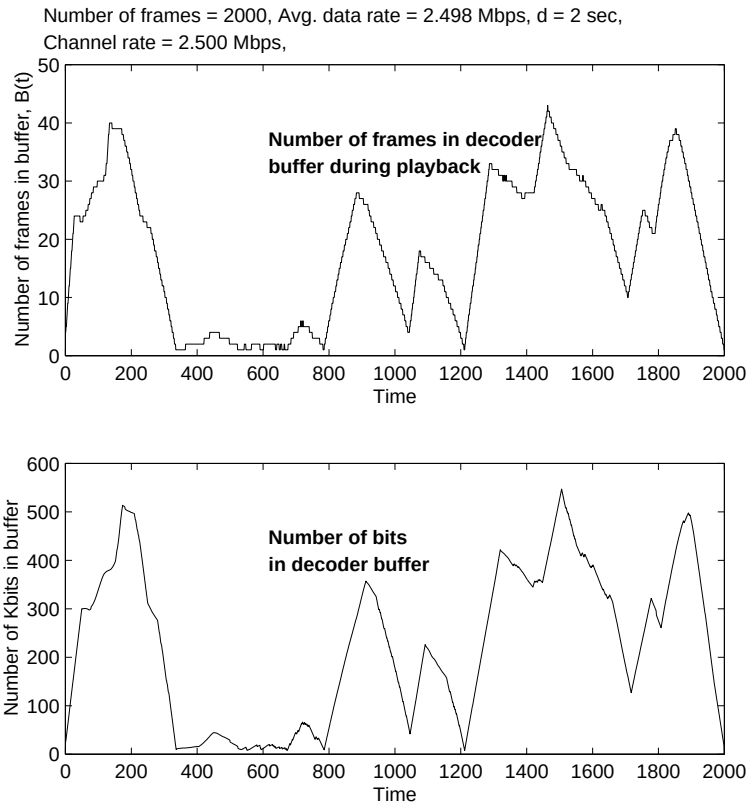

Figure 3.5: Trace of client buffer size in number of frames and bits.

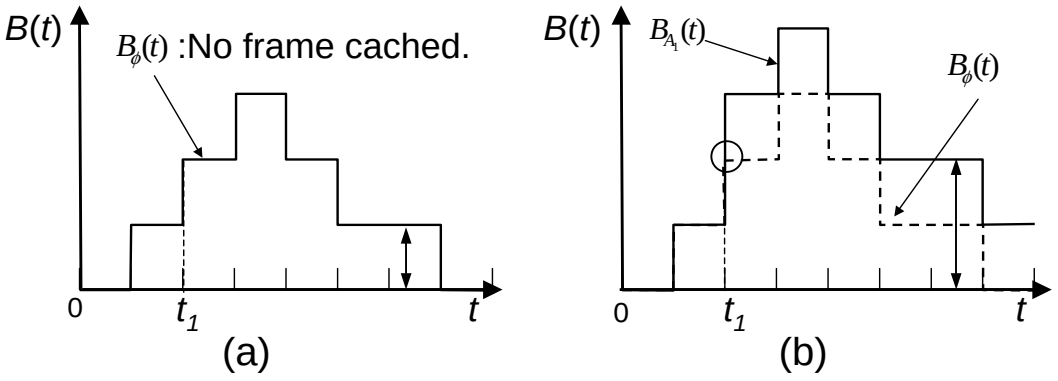

Figure 3.6: (a): $B_\phi(t)$, no frame is cached. (b): $B_{\mathcal{A}_1}(t)$, frame $F(t_1)$ is cached. These figures illustrated that by caching one frame $F(t_1)$, the buffer trace can be "lifted" for $t \geq t_1$, when there is no buffer size limitation.

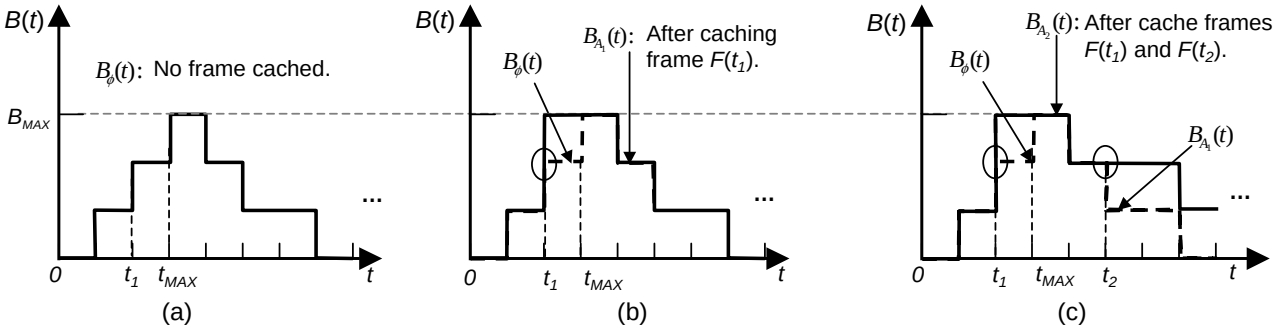

Figure 3.7: The client buffer size limitation is B_{MAX}. (a): $B_\phi(t)$, no frame is cached. (b): $B_{\mathcal{A}_1}(t)$, after $F(t_1)$ is cached. (c): $B_{\mathcal{A}_2}(t)$, after both $F(t_1)$ and $F(t_2)$ are cached. With the buffer size limitation, the buffer trace after caching frames will not follow that in Fig. 3.6. (b) shows that caching frame $F(t_1)$ only increase $B_{\mathcal{A}_1}(t)$ between $t_1 \leq t \leq t_{max}$. (c) shows that caching frame $F(t_2)$ can lift buffer trace for all $t \geq t_2$ because there is no maxima point after t_2.

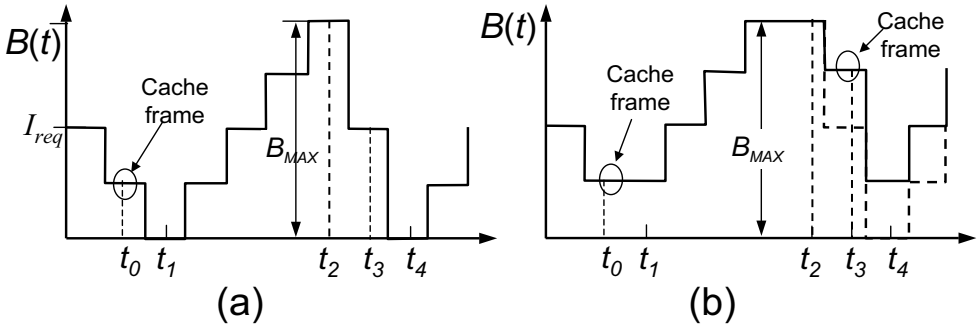

Figure 3.8: (a): Trace where only I_{req} is cached and troughs occur at time t_1 and t_4. (b): After SCQ caching. First select frames before t_1, but t_4 still remains the same, due to the maximum peak at t_2, drawn in dotted line. Next select frames within $[t_2, t_4]$ to increase robustness for t_4.

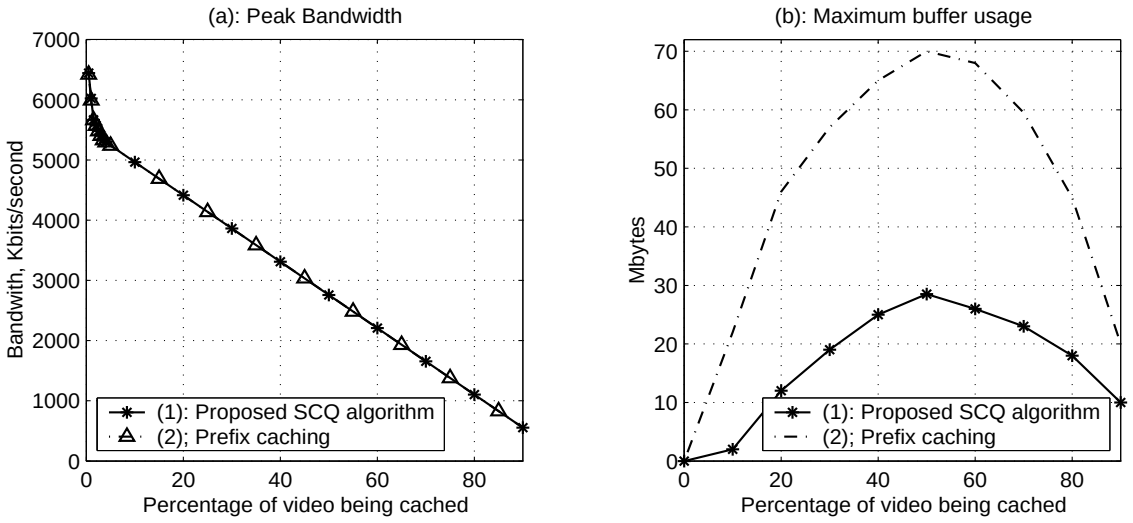

Figure 3.9: (a) The bandwidth ($C_r(\mathcal{A})$, see (3.7)) that has to be reserved, v.s. the percentage of the video been cached. (a): Both the proposed SCQ and prefix caching can reduce $C_r(\mathcal{A})$ similarly as more portion of the video has been cached. (b): The maximum buffer size, B_{max}, required at the client to achieve the caching performance in (a).

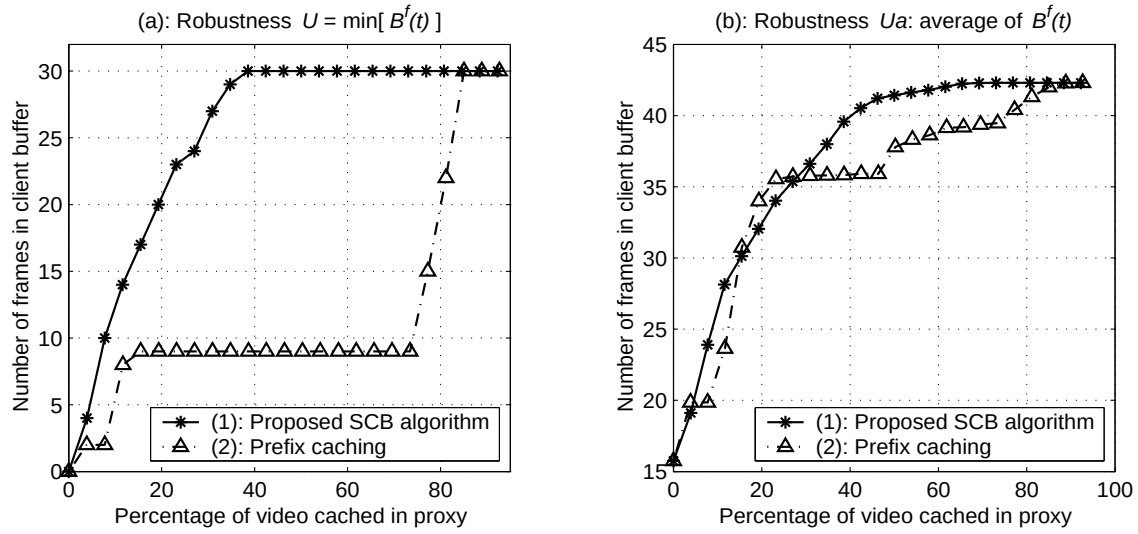

Figure 3.10: Robustness vs the percentage of the video been cached, using SCB and prefix caching methods. (a) Robustness U defined in (3.17). (b) Robustness U_a defined in (3.18).

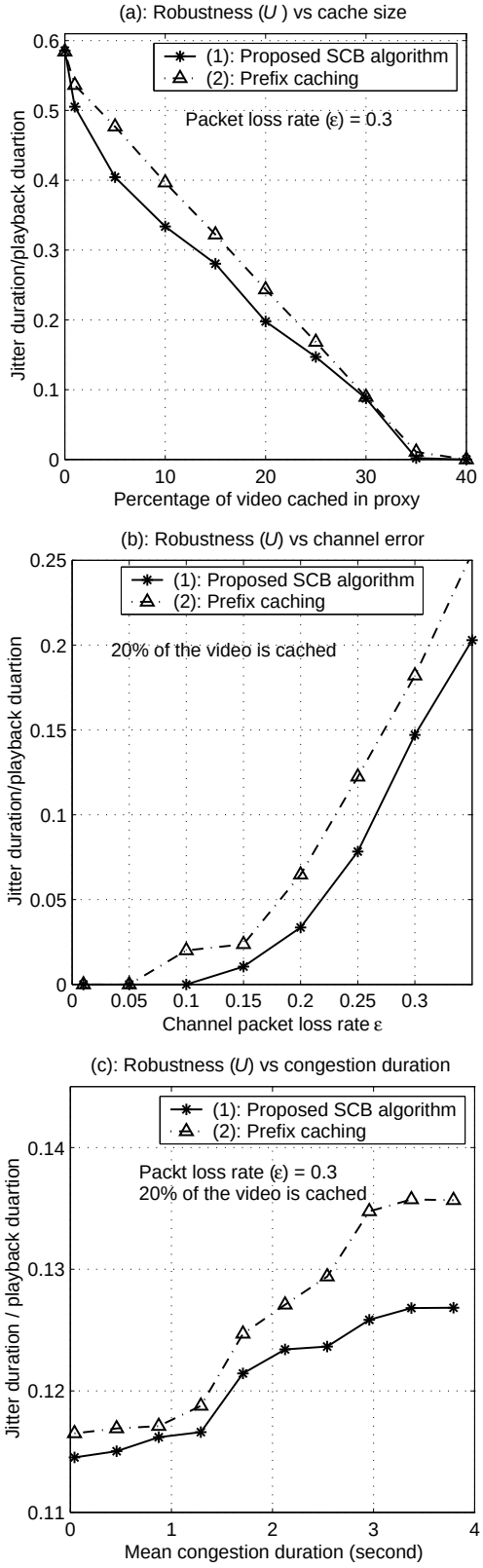

Figure 3.11: Robustness verification, with 1000 realizations used in each case. (a): T_J/T_V vs percentage of the video being cached. (b): T_J/T_V vs channel error ϵ, when 20% of the video is cached. (c): T_J/T_V vs average channel congestion duration d_c.

Chapter 4

Video Compression with Rate Control for Video Storage on Disk Based Video Servers

4.1 Introduction

Video-On-Demand (VOD) services have been studied in the past few years and may soon become popular, as recording, storage and transmission of video data becomes less costly. The two challenging problems in a VOD system are video data transmission and disk storage. The output bit rate after compression is usually Variable Bit Rate (VBR),[1] but common transmission channels are based on the Constant Bit Rate (CBR) mode where the transmission bandwidth is fixed. Thus, transmission of VBR video data through a CBR channel may generate hiccups (i.e. frame loss) [10, 46, 32, 55].

[1]The variable nature of the bit rate per frame comes from the fact that frames, when compressed to achieve a specific visual quality, require a different number of bits depending on such factors as the number of objects in the frame, the motion, the proportion of textured areas to flat areas, etc.

In this chapter, we first address the disk storage issues and, in particular, we study how video data can be encoded to achieve more efficient transmission. This will lead us to the rate control algorithms optimized for specific disk placement strategies.

Since the volume of video data is large, large capacity and high-speed hard disks are commonly used to store it. These modern disks have very high transfer bandwidth, and thus it is possible for a server to provide continuous video display to several users simultaneously. The disk drive can be multiplexed among several displays by providing *Round Robin Service* [30].

An example is shown in Fig.-4.1. A number of users ($W, X, ...Z$) are served by the server. Each user is allocated a "time slot" (T_{slot}) during one round interval (T_{round}) to receive a block of data from the server. This block (e.g., W_i) of data should contain sufficient number of data (frames) for the user to display video until the next block arrives, the required display time is also equal to T_{round}. If there is not sufficient data in that block, jitters will occur during the playback at the user. For a fair service, each user gets the same amount of compressed data in each service round.

The video data blocks have to be retrieved from disks before the server send it to the network channels. When the disk is large (which is typical for the purpose of storing large volume of video data), the time spent to find a particular video data block on the disk, referred as *disk seek-time*, can be significant [30, 7, 9, 10]. Since

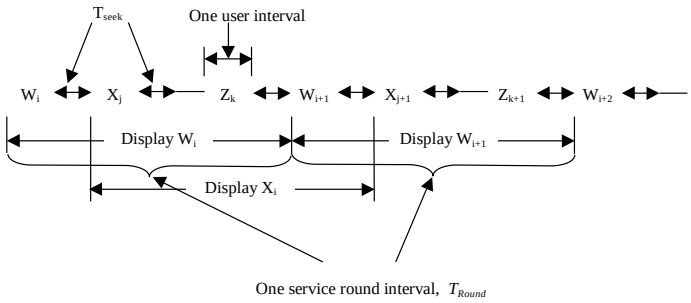

Figure 4.1: Service Round. During each service round time, T_{round}, many users can retrieve data from the server for playback. This figure shows an example of Round Robin Service, where each user is allowed to retrieve a block data from a particular time slot during each service round.

data can not be transferred from disk during the seek-time, a smaller seek-time overhead allows more data to be delivered from disk to the the users during one round T_{round} and therefore, more users can be supplied by the server concurrently. The disk seek-time also introduces another side effect: if the video blocks are placed in a random order, the seek-time can be *variable* to access different blocks, which will lead T_{round} to be variable[2] and unpredictable. If T_{round} becomes too long, there are more chance for the client to run out of frames to be displayed (where jitter occurs) before it get another data block from next service round.

Studies in [30, 7, 9, 10] show different approaches to reduce the seek-time by place the video objects on the hard disk special orders. Many of these approaches partition the disk and place the video data blocks on the disk *sequentially.* according to the display order, since the video data is likely to be requested sequentially (from

[2]We assume T_{slot} is fixed to transfer a complete data block. Otherwise if we vary T_{slot} to make T_{round} to be constant, then some data blocks may have to be delivered during more than one time slot, which means more disk seek-time has to be spent to find the *same* data block for each additional delivery

the earlier frames to later frames) for playback. The disk partition creates "regions". One region contains several video data blocks from *different* video objects, see Fig.-4.2 for an example. The data blocks from different video objects forms an interlaced fashion of placement. The reason for the "interlaced fashion" is that disk bandwidth is large enough to transfer one data block which contains enough frames to playback during the time when the disk transfer other blocks in the same region. For example, after transferring data block V_i^1 (the i^{th} data block of video object 1), the disk arm continues to move in *one direction* to retrieve other blocks in the same region. With carefully design, the disk arm takes less then T_{round} time when it reaches block V_{i+1}^1 to continue transfer next block of video object 1. Therefore the data can be transferred for different video objects during the disk arm movement showed in Fig.-4.2. It is shown that this kind of placement can reduced the overall disk seek-time when different video objects are requested simultaneously by several concurrent users. For more details of disk placement strategies, refer to [7, 9, 10]. If blocks are placed in a more restrictive sequence (e.g. zigzag mode [30]), the seek-time can be reduced to close to zero.

Another benefit from these disk placement strategies is that the disk seek-time is relatively constant for different data blocks, so that the resulting T_{round} is nearly constant, which reduces the complexity of software design of the server system and improves its performance.

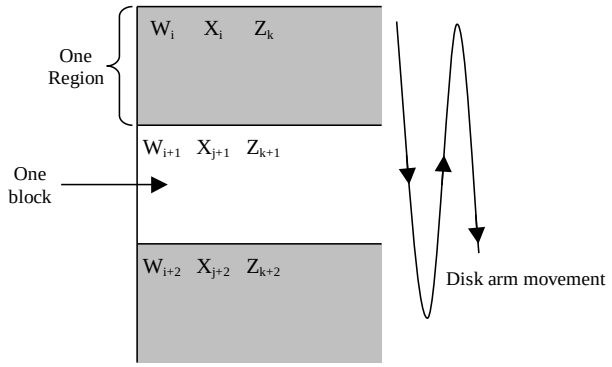

Figure 4.2: Disk placement. The video blocks are placed in an "interlaced" fashion so that disk seek-time can be reduced when multiple video objects are requested concurrently

These disk placement algorithms target the efficiency[3] of VOD servers, but they may affect the servers' ability to provide continuous display. For example, some data placement algorithms may not allow random access because the location of the data blocks has been restricted. Thus it will be more difficult than that in a pure random placement approach to reduce the *duration* of hiccups. This is because, once the maximum number of users N is set, the block size B_{size} is also fixed (transmitted during each T_{round}) [30]. Thus, after the *disk placement* algorithm has been applied, it is *not* possible for the disk arm to reach a block at an arbitrary location, because the disk arm moves in a single direction (either towards the edge or the inside). If a block can not be displayed for T_{round} long, the result is that a user *must wait for the next service round* to get the next block. The server may have to "pause" service to other users by spending extra time on one particular client to reduce its hiccups.

[3]i.e., they try to maximize the number of users that can be served simultaneously.

Studies [10, 46] show different strategies to reduce the hiccups, such as restricting the number of concurrent users based on the Quality-of-Service parameters. Other approaches involve making a decision on whether to admit a new user based on the probability of hiccups if that user is admitted. Some of these admission algorithms may be too restrictive, and while they may prevent hiccups, they may also waste bandwidth during some service rounds (when all the users request small size blocks). Conversely, a less demanding admission policy may result in more hiccups.

In this chapter, we tackle this problem from the encoder view point. We encode the video data with the given constraints set (B_{size}, T_f, N) to guarantee continuous display given a particular disk placement specification. Admission of a new user will be simple, we just need to check to see if the total number of users exceeds the maximum number allowed (N). Since N is used to calculate the bandwidth per user applied in the rate control optimization, we can guarantee that no hiccups will occur if the number of users is less than N. This approach could also be incorporated with other algorithms mentioned above, if we allow limited hiccups during the display. Note that an easy way to achieve this is to compress the video with restrict Constant-Bit-Rate (CBR), so that each data block with same size is guaranteed to supply a constant period of time, e.g, T_{round}. However, typically a CBR video has lower quality (measured in Peak Signal to Noise Ratio, PSNR) then a Variable-Bit-Rate (VBR) video for the rate budget, and VBR video is generally supported by many current standards, such as MPEG-1, -2 [52], H.263 [34], MPEG-4 [3]. In this

paper we code the video in VBR formate to achieve better quality, and propose a rate-distortion based algorithm for video compression under the constraints of disk placement strategies, which selecting different quatizers for each frame to find the optimal bit allocate for each frame. The experimental results shows that the video quality is improved by 0.5dB to 1.5dB compared the compression method without rate control.

The organization of this chapter is as follows. Section II presents the formulation of the problem. Section III describes a *Multiple Lagrange Multiplier Algorithm* to obtain the optimal solution, and Section IV provides the experimental results and conclusions. Our results show that the overall PSNR with rate control can be improved by 0.5 to 1.5 dB as compared to not using rate control under the constraints of continuous displaying.

4.2 Problem Formulation

User buffer constraints: We assume that the video frame rate is constant. In a real-time video transmission system, the end-to-end delay interval must be constant, say ΔT seconds. Thus a frame read from disk at time t, must arrive at the decoder (user) before $t + \Delta T$. As for a VOD system, all the video data is available before transmission, the size of the user buffer and initial latency should be small (i.e., we assume the user can only pre-fetch a smaller number of frames, f_p, before playback). Real-time transmission constraints are still applicable in VOD systems, because once

T_f	Period during which a frame is displayed, e.g. $1/30 second$
f_p	Number of pre-fetched frames
T_p	Period during which the pre-fetched frames are displayed, $T_p = f_p \times T_f$
T_{round}	Turn-around-time for one service round
F_i	i-th frame
B_{disk}	Disk bandwidth (bits/second)
N	Maximum number of users
B_{user}	The average bandwidth per user $B_{user} = B_{disk}/N$
C_{acc}	Accumulated channel rate
R_{acc}	Accumulated frame rate
Buf_{max}	User buffer size
C_k	Channel rate at time $t = kT_f$
$x(i)$	Quantization step for frame i
$R_{x(i)}(i)$	Number of bits for frame i coded with quantization step $x(i)$
$D_{x(i)}(i)$	Distortion of frame i coded with quantization step $x(i)$
N_f	Total number of frames
B_{size}	Block size in disk placement algorithms
R_N	Number of total service rounds
$N_B(i)$	Number of frames in block i

Table 4.1: Notation used in this chapter

we start displaying frames, we need to continuously transmit frames, and the number of frames that can be stored in the decoder buffer is limited (e.g. the whole sequence cannot be stored in the buffer).

End-to-end Delay: We assume the frame rate is *30 frames/sec* (it can be other number depending on the video object). The pre-fetched f_p frames will be displayed for $T_p = f_p/30$ *seconds*. If a frame is scheduled to be displayed at time t during the playback, it should arrive at the user end before $t + T_p$.

Rate constraint: The delay and buffer constraints can be converted into rate constraints, which the encoder has to meet to prevent hiccups. We assume that T_f is the period each frame can be displayed (typically, $1/30 second$), which is also the "time unit" we will use in this paper. Each frame is labeled with index $i, i = (1, 2, 3 \ldots)$, and if we start display at time $t = 0$, the frame F_i will be scheduled to display at time $t = i \times T_f$. R_i is the number of bits of frame F_i.

The channel rate is the disk bandwidth allocated to each user after time multiplexing. According to Table-1, each user will get $B_{user} = B_{disk}/N(bits/sec)$ of bandwidth on average. That is, each user will receive bandwidth B_{disk} during a fraction T_{round}/N of each service round. The constraint for *no buffer underflow* is that any frame F_i must arrive at the user no later than $t = i \times T_f$. This requires that the channel have enough capacity to transmit all the frames $(F_1, F_2 \ldots F_i)$ to

the user before time $t = i \times T_f$. Denote $C(k)$ as the disk bandwidth (allocated to the particular user) at time $t = kT_f$, we get

$$\sum_{k=1}^{i} R_{x(k)}(k) \leq \sum_{k=1}^{i+f_p} C(k), \qquad i = 1, 2, 3, ..., N_f \tag{4.1}$$

where $R_{x(k)}(k)$ is the size of k-th frame encoded with quantization step $x(k)$. In this paper we use the video sequences compressed using MPEG-1 with intra-mode, which means each frame is coded independently The frame size and quality can be adjust by the quantization step size $x(k)$ (chosen from an available quantization step set) used for that frame: a larger $x(k)$ leads to lower bit rate of the frame (smaller $R_{x(k)}(k)$) with poor quality; while a smaller $x(k)$ leads to a larger $R_{x(k)}(k)$ with higher quality.

We can use *Accumulated channel rate (C_{acc})* and *accumulated frame rate (R_{acc})* to describe the problem more clearly. We can re-write the (4.1) as (with frames pre-fetching):

$$R_{acc}(i) = \sum_{k=1}^{i} R_{x(i)}(i), \qquad C_{acc}(i) = \sum_{k=1}^{i} C_k, \tag{4.2}$$

$$R_{acc}(i) \leq C_{acc}(i) + \sum_{k=1}^{f_p} C(k), \tag{4.3}$$

$$where \;\; C(k) = \begin{cases} B_{disk}, & nT_{round} \leq k < nT_{round} + T_{user} \\ 0, & otherwise \end{cases}$$

Here R_{acc} is simply the accumulated bits of all the frames that have been sent. $C(k)$ equals B_{disk} while the server is transmitting data to that user; it is zero while the server is serving other users. We assume there are no constraints introduced by the network bandwidth here. Fig.-4.3 shows and example of R_{acc} and C_{acc}. For a continuous playback, the curve of R_{acc} must below C_{acc}, otherwise hiccups will occur during the playback at the time when R_{acc} is larger than C_{acc}.

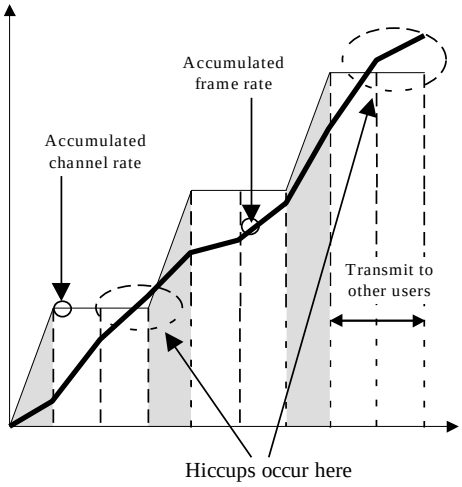

Figure 4.3: Accumulated channel rate and frame rate.

If different video sequences are stored on the disk, and different users request different sequences randomly, buffer underflow can be avoided for all clients if all the video sequence is encoded with rate control set by (4.1). This is because that if each sequence is encoded with rate constraints, and each of them will meet the requirement of no buffer underflow. These constraints are still met for each user even if the sequences are requested simultaneously.

In real-time playback, lost frames will cause visual distortion. For our scenario, hiccups mean the user has to wait for the next frame, while the current frame is

frozen on the screen (or other error concealment can be applied). Thus, it may be preferable to encode a frame with fewer bits if that allows us to avoid hiccups. Although the distortion of this frame would be increased, it is better than displaying nothing in certain cases. In the cost function below, we do not allow any frame loss in our formulation[4]. The quantization step could be chosen at the encoder end before the video streams are stored onto the disk. The problem can be formalized as follows.

Given a set of constraints (as in [55]), how do we choose the quantization step size for each frame while minimizing the total distortion. To encode N_f frames, using a given set Q of M admissible quantizers, such that, for each choice of quantizer $j = x(i)$ for a given block i, we incur a distortion cost $D_{x(i)}(i)$ while requiring a certain rate $R_{x(i)}(i)$. The objective is to *find the optimal quantizer choices* $x^* \in \chi = Q^N$, *for a given channel rate* C_k *as in (4.3), such that:*

$$x^*(1, ..., N_f) = \text{argmin} \sum_{i=1}^{N_f} D_{x(i)}(i) \tag{4.4}$$

subject to the constraint set (4.1) or (4.3). We will solve this problem using *multiple* Lagrange Multipliers in the next section.

[4]It is possible to develop other cost functions which may take account frame losses. This is beyond the scope of this chapter.

4.3 Optimization based on Multiple Lagrange Multipliers

Using Lagrangian optimization for rate control under multiple rate constraints was previously studied in [55, 12]. In that approach, the constrained optimization problem above is equivalent to the unconstrained problem derived by introducing a non-negative Lagrange multiplier λ_i associated with each constraint in (4.2). The optimization formulation then becomes: *find the quantizer choice* x^* *at the time* $t_i = iT_f$ *such that:*

$$x^*(1, N_f) = \arg\min_x \sum_{i=1}^{N_f} D_{x(i)}(i) + \sum_{j=1}^{N_f} \lambda_j \Big(\sum_{i=1}^{j} R_{x(i)}(i) \Big), \tag{4.5}$$

We introduce N_f Lagrange multipliers to replace the N_f constraints in equation (4.3). To find the *optimal* quantizer set $x^*(1, N_f)$ is the same as to to find the appropriate multipliers $\{\lambda_i\}$ to meet the constraints. From [32], we can introduce another set of multipliers $\lambda_i' = \sum_{j=i}^{N_f} \lambda_j, (i = 1, 2, ...N_f)$ to rearrange (4.5) as:

$$x^*(1, N_f) = \arg\min_x \sum_{i=1}^{N_f} (D_{x(i)}(i) + \lambda_i' R_{x(i)}(i)), \tag{4.6}$$

Finding the solution for (4.5) is equivalent to finding the appropriate non-negative values of the set $\{\lambda_i'\}$. Define $J_i(\lambda_i', x(i))$, the cost for frame i, as:

$$J_i(\lambda_i', x(i)) = D_{x(i)}(i) + \lambda_i' R_{x(i)}(i), \tag{4.7}$$

If we use intra-frame mode, the quantizer for each frame can be chosen independently while minimizing the cost for each block $J_i(\lambda_i', x(i))$ as:

$$x^*(i) = \arg \min_{x(i) \in Q} J_i(\lambda_i', x(i)), \quad \forall i \in \{1, 2, ..., N_f\} \tag{4.8}$$

In [55, 32] a similar problem is solved by iteratively increasing the lower bounds on the multipliers, defined as $\{\Lambda_i'\}$, such that the violation of rate constraints can be prevented, and adjusting the values of $\{\lambda_i'\}$ until an optimal bit allocation, where none of the constraints is violated, is found. The details of the search for these multiple Lagrange multipliers can be found in [55, 32, 81]. Here we outline the basic procedures.

Step 1: Initially the quantizer choices $\hat{x} = \{x(1), x(2), ..., x(N_f)\}$ are obtained by using a single Lagrange multiplier λ_{N_f}' for all the frames in (4.8), subject to only one constraint: $\sum_{k=1}^{N_f} R_{i,x(i)} \leq \sum_{k=1}^{N_f + f_p} C_k$.

Step 2: If $\hat{x}$ is such that all rate constraints in (4.1) are met, then $\hat{x}$ is the optimal solution x^* for problem (4.4). Otherwise, assume that frame v is the last frame which violates the rate constraint, that is, $v < N_f$ and no other frame between frame $v+1$ and frame N_f violates the rate constraint. Find the minimum value of Lagrange multiplier $\Lambda_v' = \min \lambda_v'$ for the video stream from frame 1 to frame v which prevents violation of the rate constraint: $\sum_{k=1}^{v} R_{i,x(i)} \leq \sum_{k=1}^{v + f_p} C_k$.

Step 3: Find the quantizer choices $\hat{x} = \{x(1), x(2), ..., x(N_f)\}$ as in Step 1 except that the Lagrangian multiplier for the video streams from frame 1 to frame v is lower-bounded by Λ'_v as $\lambda'_v \Leftarrow \max(\Lambda'_v, \lambda'_v)$.

Step 4: Go to Step 2. Repeat until all the rate constraints in (4.1) are met.

4.4 Experimental Results

In order to test our proposed algorithms, we simulated the transmission behavior with and without the rate control. We use 5000 and 10,000 frames from the movie "Mission Impossible" for our simulation. Each frame was encoded in intra-frame mode and we use 7 different quantization steps encoded by JPEG, thus generating 7 source streams with different rate-distortion performance. Each source stream uses a fixed quantizer. We test with different parameters for B_{disk}, T_{round} and N.

Fig. 4.4 shows the accumulated channel and frame rate (with and without rate control). The rates are added up from the time the video transmission starts. Curve (1) is accumulated channel rate ($C_{(acc)}$) which is the upper bound of the frame rate. The other curves (2-3) are accumulated frame rate ($R_{(acc)}$). The hiccups will occur if $R_{(acc)}$ exceeds the $C_{(acc)}$. Among curves (2-3), curve (2) is closest to the bound (1), which is based on rate control processing (Lagrange iteration). Curve (3,4) use fixed quantization steps, with a smaller quantizer step size for curve (4) (high frame rate, low distortion), and a larger one for curve (3). Those two quantization step are the closest two consecutive steps size of all the available steps.

The channel rate is the disk bandwidth, the block size is a typical size from certain disk placement algorithms. These two parameters decide the shape of curve (1), the upper frame rate bound. It shows that with a smaller fixed quantization step, there are more hiccups, while with a larger one, the channel capacity is wasted.

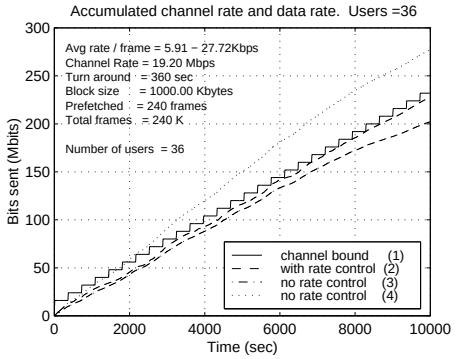

Figure 4.4: Accumulated channel rate and frame rate

After applying the rate control over the whole sequence, a different quantization step can be chosen for each frame, and the accumulated frame rate can be set very close to the channel bound without exceeding it. We compared the PSNR with the distortion using rate controls with the frames of fixed quantization (its rate also not exceed the channel bound). As there is no common methods to measure the distortion (neither measured by PSNR or by some perceptual measurement) for a "lost" frame, the comparison is made based on that no hiccups occur for any choice of the quantizers (with or without rate control). Based on our experimental result, we have about 0.5 to 1.5 dB for overall frame sequences as showed in Fig.-4.5. Of

course, if there are less available quantization steps, we can get larger PSNR gain, and small PSNR gain vice versa[5].

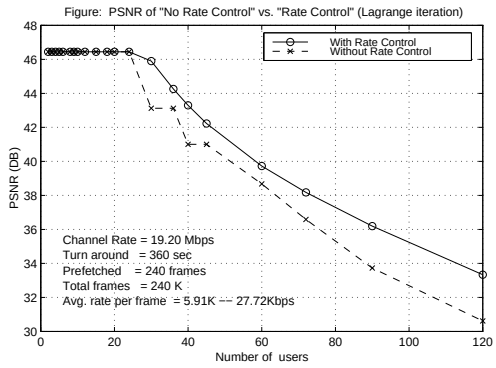

Figure 4.5: PSNR of the video sequence. When the maximum number of the concurrent user can be supported by the server increases, the bandwidth allocated to each user decreases, therefore the video bit rate has to be reduced which leads to poor quality. Using rate control based compression can improve the overall PSNR (averaging over the frames) by 0.5 to 1.5 dB compared to the scheme without any rate control (a uniform quantizer is used for all frames).

For the other choices of uniform quantizers which leads to different quality and bit rate of the compressed video, we show the total hiccups and average user waiting time during the hiccups (hiccup duration) in Fig-4.6. The results shows that when the number of designed maximum user increases, more hiccups will occur for the video coded with a particular quantizer, and the average waiting time (during which a frame has to be frozen on the screen until the next video block arrives) also increases. This is the drawback of using a uniform quantizer for compression. Note that with rate control, the hiccups can be guaranteed to be avoided for a given maximum number of concurrent users.

[5]If the set of available quantization scales is small than it will be more likely that a solution which does not violate the rate constraints is far below the bound.

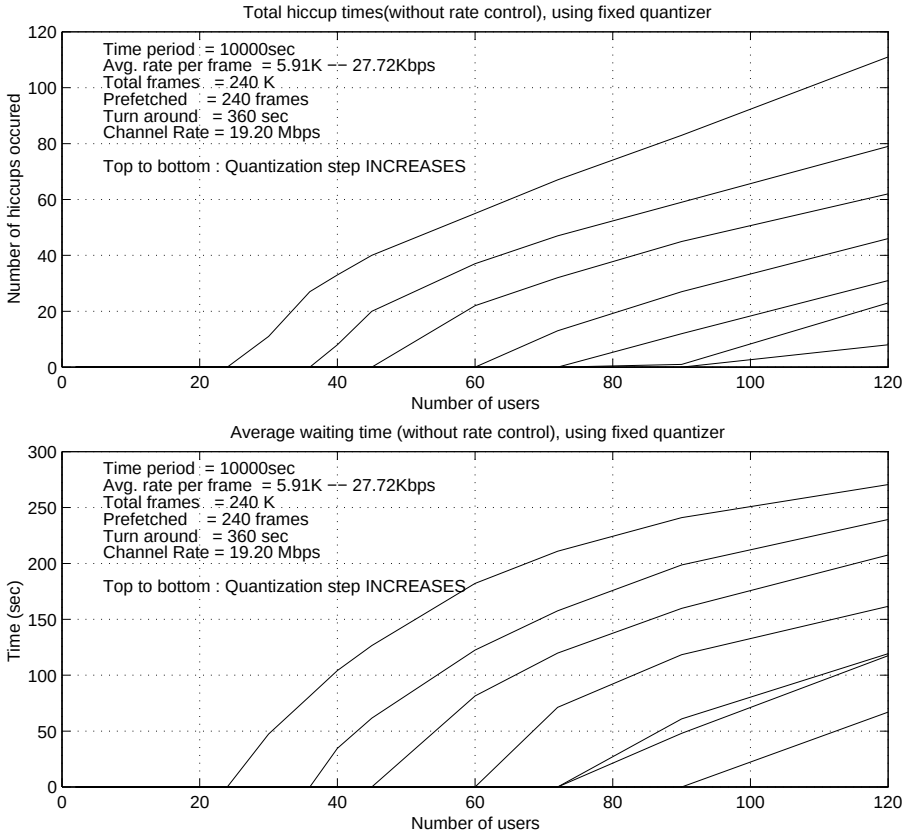

Figure 4.6: Average waiting time and hiccups without rate control

4.5 Conclusions

In this paper we analysis the impact on continuous playback of the video stream by some particular video disk storage strategies. The disk storage strategies are designed to improve the server throughput by reducing the disk seek time (for finding a particular video data block on the disk) so that more users can be served currently. We found that certain disk storage strategies imply some restrictions the bit rate of the VBR video stream to meet their data retrieval patterns. We translated those restrictions into rate constraints and the cost functions for our proposed rate control

based compression algorithm, Basically, the proposed algorithm selects the proper quantizers for different frames in the video to maximize the overall (averaging over the frames) video quality (measure in PSNR) while the rate constraints are not violated. Our experimental results show the proposed scheme has 0.5 to 1.5 dB quality improvement compared to the compression scheme without any rate control, which uses uniform quantizer for all frames in a video object.

Chapter 5

Conclusions

In this thesis several novel algorithms are presented for streaming media services, namely, the delivery scheduling algorithm (ERDBS) for scalable streaming media over best-effort networks; the two selective caching algorithm for video proxy caching in QoS networks (SCQ algorithm) and best-effort networks (SCB algorithm); a rate-control based compression technique under the constraint of video server disk placement.

The thesis discussed each algorithm in details and shows that there is performance improvement by using these algorithms. For example, EDBS algorithm can improve the playback quality of streaming media compared to the traditional delivery methods. The SCQ video caching algorithm can reduce the QoS network cost maximumly while a smaller receiver buffer size is required and the caching space is limited (only part of the video can be cached); the SCB video caching algorithm can increase the robustness of continues playback of streaming video against poor

network condition in best-effort networks (under the same constraints as QoS networks, such as limited cache space and receiver buffer size). Finally, using rate control based video compression with certain video server disk placement strategy can secure the benefit of the particular disk placement strategy and gaurante the conitnues playback at the client.

Reference List

[1] ISO/IEC JTC 1/SC 29/WG 1, JPEG-2000 Image Coding System, (WG1N390 REV).

[2] Resource reservation protocol (RSVP) – version 1 functional specification, RFC 2205, proposed standard, September 1997.

[3] ISO/IEC JTC1/SC29/WG11, MPEG-4 version 2 visual working draft rev. 3.0, N2202, March 1998.

[4] Real time streaming protocol (RTSP), proposed standard, RFC 2326, April 1998.

[5] S. Acharya. *Techniques for improving multimedia communication over wide area networks.* PhD thesis, Cornell University, 1999.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. On the implications of Zipf's law for web caching. In *Proc. of IEEE Infocomm*, 1999.

[7] D. W. Brubeck and L. A. Rowe. Hierarchical storage management in a distributed VOD system. *IEEE, Multimedia*, pages 37–47, Fall 1996.

[8] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems*, pages 193–206, Dec 1997.

[9] E. Chang and H. Garcia-Molina. Reducing initial latency in media servers. *IEEE, Multimedia*, pages 50–61, Fall 1997.

[10] E. Chang and A. Zakhor. Disk-based storage for scalable video. *IEEE Trans. On Circuits and Systems for Video Technology*, 7(5):758–770, Oct 1997.

[11] A. Chankhunthod, P. B. Danzig, C. Neerdals, M. F. Schwartz, and K. J. Worrell. A hierarchical Internet object cache. In *USENIX Tech. Conf.*, 1996.

[12] J-J. Chen and D. W. Lin. Optimal bit allocation for coding of video signals over ATM networks. *IEEE JSAC*, 15:1002–1015, Aug. 1997.

[13] Y. S. Chen and P. Chong. Mathematical modeling of empirical laws in computer applications: A case study. *Comput. Math. Applicat.*, pages 77–87, October 1992.

[14] Wu chi Feng, F. Jahanian, and S. Sechrest. An optimal bandwidth strategy for the delivery of compressed prerecorded video. *ACM/Springer-Verlag Multimedia Systems Journal*, 5(5), Sept. 1997.

[15] P. A. Chou and Z. Miao. Rate-distortion optimized sender-driven streaming over best-effort networks. In *IEEE Workshop on Multimedia Signal Processing*, pages 587–592, Cannes, France, October 2001.

[16] P. A. Chou and Z. Miao. Rate-distortion optimized streaming of packetized media. Technical Report MSR-TR-2001-35, Microsoft Research Center, February 2001.

[17] P. A. Chou and Z. Miao. Rate-distortion optimized streaming of packetized media. *IEEE Multimedia*, February 2001. Submitted.

[18] P. A. Chou, A. E. Mohr, A. Wang, and S. Mehrotra. FEC and pseudo-ARQ for receiver-driven layered multicast of audio and video. In *Proc. Data Compression Conf.*, Snowbird, UT, March 2000. IEEE Computer Society.

[19] P. A. Chou, A. E. Mohr, A. Wang, and S. Mehrotra. Error control for receiver-driven layered multicast of audio and video. *IEEE Transactions on Multimedia*, 2001.

[20] T. Cover and J. Thomas. *Elements of Information Theory.* John Wiley & Sons, 1991.

[21] A. Dan and D. Sitaram. Multimedia caching strategies for heterogeneous application and server environments. In *Multimedia Tools and Applications*, volume 4, pages 279–312, 1997.

[22] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proc. of ACM Multimedia*, pages 391–398, Oct. 1994.

[23] M. de Prycker. *Asynchronous Transfer Mode: Solution for Boardband ISDN.* Ellis Horwood, Chichester, England, 1991.

[24] B. Dempsey, J. Liebeherr, and A. Weaver. On retransmission-based error control for continuous media traffic in packet-switching networks. *Computer Networks and ISDN Systems Journal*, 28(5):719–736, March 1996.

[25] Cisco Caching Engine. http://www.cisco.com/warp/public/751/cache.

[26] N. Farvardin. A study of vector quantization for noisy channels. *IEEE Trans. on Information Theory*, 26:799–809, July 1990.

[27] N. Farvardin. A study of vector quantization for noisy channels. *IEEE Trans. on Information Theory*, 26:799–809, July 1990.

[28] G.D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61:268–278, Mar. 1973.

[29] M. W. Garrett. *Contributions Toward Real-Time Services on Packet Switched Networks*. PhD thesis, Dept. of Electrical Eng., Columbia Univ., 1993.

[30] S. Ghandeharizaheh, S. H. Kim, and C. Shahabi. On configuring a single disk continuous media server. In *Proceedings of the ACM SIGMETRICS/PERFORMANCE*, May 1995.

[31] S. Gruber, J. Rexford, and A. Basso. Protocol considerations for a prefix-caching proxy for multimedia streams. *WWW9 / Computer Networks*, 33(1-6):657–668, 2000.

[32] C. Y. Hsu, A Ortega, and M. Khansari. Rate control for robust video transmission over burst-error wireless channels. *IEEE JSAC, Special Issue On Multimedia Network Radios*, 17(5):756–773, May 1999.

[33] C.-Y. Hsu, A. Ortega, and A. Reibman. Joint selection of source and channel rate for VBR video transmission under ATM policing constraints. *IEEE Journal on Select Areas in Communications*, 15:1016–1028, Aug. 1997.

[34] ITU-T. Video coding for low bitrate communication. ITU-T Recommendation H.263; version 1, Nov. 1995, version 2, Jan. 1998.

[35] W. Jiang and A. Ortega. Multiple description coding via polyphase transform and selective quantization. In *Proc. of VCIP*, 1999.

[36] M. Vetterli K. Ramchandran. *Multiresolution Joint Source Channel Coding*. Wireless Communications:Signal Processing Perspective. Prentice Hall, 1998.

[37] M. Kamath, K. Ramamritham, and D. Towsley. Continuous media sharing in multimedia database systems. In *4th International Conference on Database Systems for Advanced Applications*, Apr. 1995.

[38] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross. Distributing layered encoded video through caches. In *Proc. of IEEE Infocom*, Anchorage, AK, USA, 2001.

[39] J. Kangasharju, Y. Kwon, A. Ortega, X. Yang, and K. Ramchandran. Implementation of optimized cache replenishment algorithms in a soft caching system. In *IEEE Signal Processing Society Workshop on Multimedia*, CA, Dec. 1998.

[40] R. Koenen. MPEG-4, multimedia for our time. *IEEE Spectrum, pages 26–34*, 1999.

[41] P.Pancha L. Xue, S. Paul and M. Ammar. Layered video multicast with retransmission (LVMR): Evaluation of error recovery schemes. In *Proc. NOSSDAV*, pages 161–172, St. Louis, MO, May 1997.

[42] M. Lucas, B. Dempsey, and A. Weaver. MESH: distributed error recovery for multimedia streams in wide-area multicast networks. In *Proc. IEEE Int. Conf. on Commun.*, volume 2, pages 1127–32, Montreal, Que., June 1997.

[43] W.-H. Ma and H.C. Du. Reducing bandwidth requirement for delivering video over wide area networks with proxy server. In *Proc. International Conference on Multimedia and Expo.*, volume 2, pages 991 –994, 2000.

[44] A. Mahanti, C. Williamson, and D. Eager. Traffic analysis of a web proxy caching hierarchy. *IEEE Network*, 14(3):16–23, May-June 2000.

[45] A. Mahanti, C. Williamson, and D. Eager. Traffic analysis of a web proxy caching hierarchy. *IEEE Network*, 14(3):16 –23, May-June 2000.

[46] D. Makaroff, G. Neufeld, and N. Hutchinson. An evaluation of vbr disk admission algorithms for continuous media file servers. In *ACM Multimedia*, Seattle, Washington, 1997.

[47] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proc. ACM/SIGCOMM*, pages 26–30, Stanford, CA, August 1996.

[48] S. McCanne, M. Vetterli, and V. Jacobson. Low-complexity video coding for receiver-driven layered multicast. *IEEE Journal on Selected Areas in Communications*, 16(6):983–1001, August 1997.

[49] Z. Miao and A. Ortega. Rate control algorithms for video storage on disk based video servers. In *Proc. of 32nd Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, November 1998.

[50] Z. Miao and A. Ortega. Proxy caching for efficient video services over the Internet. In *9th International Packet Video Workshop (PVW '99)*, New York, April 1999.

[51] Z. Miao and A. Ortega. Optimal scheduling for streaming of scalable media. In *Proc. of 34th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, Novemeber 2001.

[52] J. Mitchell, W. Pennebaker, C. Fogg, and D. LeGall. *MPEG Video Compression Standard*. Chapman and Hall, 1996.

[53] Real Networks. http://www.real.com.

[54] J. Nussbaumer, B. Patel, F. Schaffa, and J. Sterbenz. Networking requirements for interactive video on demand. *IEEE Journal on Selected Areas in Communications*, 13(5):779–787, 1995.

[55] A. Ortega. Optimal rate allocation under multiple rate constraints. In *Data Compression Conference*, Snowbird, Utah, Mar. 1996.

[56] A. Ortega, F. Carignano, S. Ayer, and M. Vetterli. Soft caching: Web cache management for images. In *IEEE Signal Processing Society Workshop on Multimedia*, Princeton, NJ, June 1997.

[57] A. Ortega and K. Ramchandran. Rate-distortion methods for image and video compression. *IEEE Signal Processing Magazine*, 15(6):23–50, Nov 1998.

[58] Soft Caching Project Page:. http://sipi.usc.edu/ortega/ softcaching/.

[59] C. Papadopoulos and G. Parulkar. Retransmission-based error control for continuous media applications. In *Proc. NOSSDAV*, pages 5–12, April 1996.

[60] C. Papadopoulos and G. Parulkar. Retransmission-based error control for continuous media applications. In *Proc. NOSSDAV*, pages 5–12, April 1996.

[61] V. Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Trans. on Networking*, 7(3):277–292, June 1999.

[62] S. Pejhan, M. Schwartz, and D. Anastassiou. Error control using retransmission schemes in multicast transport protocols for real-time media. *IEEE/ACM Transactions on Networking*, 4(3):413–427, June 1996.

[63] W. Pennebaker and J. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[64] M. Podolsky, S. McCanne, and M. Vetterli. Soft ARQ for layered streaming media. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology, Special Issue on Multimedia Signal Processing, Kluwer Academic Publishers*, 2001, to appear.

[65] H. Radha, Y. Chen, K. Parthasarathy, and R. Cohen. Scalable Internet video using MPEG-4. *Signal Processing: Image Communication, 15 p.p. 95-126*, 1999.

[66] A. R. Reibman and B. G. Haskell. Constraints on variable bit-rate video for ATM networks. *IEEE Trans. on Circ. and Sys.*, 2:361–372, Dec. 1992.

[67] M. Reisslein, F. Hartanto, and K. W. Ross. Interactive video streaming with proxy servers. In *Proc. of First International Workshop on Intelligent Multimedia Computing and Networking (IMMCN)*, Atlantic City, NJ, Feb. 2000.

[68] M. Reisslein and K.W. Ross. Join-the-shortest-queue prefetching for vbr encoded video on demand. In *1997 International Conference on Networking Protocols*, Atlanta, October 1999.

[69] R. Rejaie, M. Handley, and D. Estrin. Quality adaptation for congestion controlled video playback over the Internet. In *Proc of ACM SIGCOMM '99*, Cambridge, MA., Sept. 1999.

[70] R. Rejaie, M. Handley, and D. Estrin. Rap: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. In *Proc. IEEE Infocom*, March 1999.

[71] R. Rejaie, M. Handley, H. Yu, and D. Estrin. Proxy caching mechanism for multimedia playback streams in the Internet. In *Proc. of 4th Web Cache Workshop*, San Diego, CA, Mar. 1999.

[72] R. Rejaie, H. Yu, M. Handely, and D. Estrin. Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet. In *Proc. of IEEE Infocom'2000*, Tel-Aviv, Israel, March 2000.

[73] Reza Rejaie. *An End-to-End Architecture for Quality Adaptive Streaming Applications in the Internet.* PhD thesis, University of Southern California, Dec. 1999. USC-Tech Report-99-718.

[74] J. Rexford and D. Towsley. Smoothing variable-bit-rate video in an Internetwork. *IEEE/ACM Transactions on Networking*, April 1999.

[75] L. Rizzo and L. Vicisano. Replacement policies for a proxy cache. *IEEE/ACM Trans. on Networking*, 8(2):158–170, April 2000.

[76] O. Rose. Statistical properties of mpeg video traffic and their impact on traffic modeling in ATM systems. Technical Report 101, Univ. of Wuerzburg, Institute of Computer Science Research Series, Feb., 1995.

[77] S. Sahu, P. Shenoy, and D. Towsley. Design considerations for integrated proxy servers. In *Proc. of IEEE NOSSDAV'99*, pages 247–250, Basking Ridge, NJ, June 1999.

[78] J. Salehi, Z. Zhang, J. Kurose, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing. In *IEEE/ACM Trans. Networking*, September 1998.

[79] D. Saparilla, K.W. Ross, and M. Reisslein. Periodic broadcasting with vbr-encoded video. In *IEEE INFOCOMM*, 1999.

[80] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications, request for comments rfc2026, Oct. 1999.

[81] G. M. Schuster and A. K. Katsaggelos. *Rate-Distortion Based Video Compression.* Kluwer Academic Publishers, 1986.

[82] G.M. Schuster and A. K. Katsaggelos. A video compression scheme with optimal bit allocation among segmentation, motion, and residual error. *IEEE Transactions on Image Processing*, 6(11):1487 –1502, Nov 1997.

[83] S. Sen, J. Dey, J. Kurose, J.Stankovic, and D. Towsley. Cbr transmission of vbr stored video. In *SPIE Symposium on Voice Video and Data Communications*, Nov. 1997.

[84] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proc. IEEE Infocom. 99*, New York, USA, March 1999.

[85] D. N. Serpanos, G. Karakostas, and W. H. Wolf. Effective caching of web objects using Zipf's law. In *Proc. of IEEE International Conference on Multimedia and Expo, ICME*, volume 2, pages 727 –730, 2000.

[86] C. E. Shannon. A mathematical theory of communication. *Bell System Tech. Journal*, 27:379–423, 1948.

[87] J. Shim, P. Scheuermann, and R. Vingrale. A case for delay-conscious caching of web documents. In *Proc. Intl. WWW Conf*, Santa Clara, CA, Apr. 1997.

[88] J. Shim, P. Scheuermann, and R. Vingrale. Proxy cache algorithms: design, implementation, and performance. *IEEE Trans. on Knowledge and Data Engineering*, 11(4):549–562, July-Aug 1999.

[89] R. Singh and A. Ortega. Erasure recovery in predictive coding environments using multiple description coding. In *Proc. of MMSP*, 1999.

[90] G. Sullivan and T. Wiegand. Rate-distortion optimization for video compression. *IEEE Signal Processing Magazine*, pages 74–90, Nov. 1998.

[91] R. Tewari, H. Vin, A. Dan, and D. Sitaram. Resource-based caching for web servers. In *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[92] R. Tewari, H. Vin, A. Dan, and D. Sitaram. Resourse based caching for web servers. In *SPIE/ACM Conference on Multimedia Computing and Networks*, San Jose, CA, USA, 1998.

[93] M. Vishwanath and P. Chou. An efficient algorithm for hierarchical compression of video. In *IEEE International Conference in Image Processing*, Nov. 1994.

[94] J. Wang. A survey of web caching schemes for the Internet. In *ACM Computer Communication Review*, volume 29(5), pages 36–46, Oct. 1999.

[95] Y. Wang, Z. L. Zhang, D. Du, and D. Su. A network conscious approach to end-to-end video delivery over wide area networks using proxy servers. In *IEEE Infocom*, Apr. 1998.

[96] G. Abdulla S. Williams, M. Abrams, and S. Patel. Removal policies in network caches for world-wide web documents. In *Proc. of ACM SIGCOMM'96*, Stanford, CA, Aug. 1996.

[97] X. Xu, A. Myers, H. Zhang, and R. Yavatkar. Resilient multicast support for continuous-media applications. In *Proc. NOSSDAV*, pages 183–194, St. Louis, MO, May 1997.

[98] X. Yang and K. Ramchandran. An optimal and efficient soft caching algorithm for network image retrieval. In *Proc. of ICIP*, Chicago, IL, Oct. 1998.

[99] Z.-L Zhang, S. Nelakuditi, R. Aggarwa, and R. P. Tsang. Efficient server selective frame discard algorithms for stored video delivery over resource constrained networks. In *Proc. IEEE INFOCOM'99*, NYC, March 1999.

[100] Z.-L Zhang, S. Nelakuditi, R. Aggarwa, and R. P. Tsang. Efficient server selective frame discard algorithms for stored video delivery over resource constrained networks. *Journal of Real-Time Imaging*, 2000.

[101] Z.-L. Zhang, Y. Wang, D. H. C. Du, and D. Su. Video staging: A proxy-server-based approach to end-to-end video delivery over wide-area networks. *IEEE Trans. on Networking*, 8(4):429–442, Aug. 2000.

[102] G. Zipf. *Human Behavior and the Principle of Least Effort.* Addison-Wesley, 1949.